Learning SQLite for iOS

The ability to use SQLite with iOS provides a great opportunity to build amazing apps. Apple's iOS SDK provides native support for SQLite databases. This combination offers the potential to create powerful, data-persistent applications.

This book starts with the architecture of SQLite database and introduces you to the concepts of SQL. You will find yourself equipped to design your own database system, administer it, and maintain it. Further, you will learn how to operate your SQLite databases smoothly using SQL commands.

You will be able to extend the functionality of SQLite by using its vast arsenal of C API calls to build some interesting, exciting, new, and intelligent data-driven applications. Understand how Xcode, HTML5, and PhoneGap can be used to build a cross-platform modern app which can benefit from all these technologies—all through creating a complete, customizable application skeleton that you can build on for your own apps.

Who this book is written for
This book is intended for those who want to learn about SQLite and how to develop apps in Swift or HTML5 using SQLite. Whether you are an expert Objective-C programmer or new to this platform, you'll learn quickly, grasping the code in real-world apps to use Swift.

What you will learn from this book
- Learn the background of SQL and SQLite for mobile development, its statements, and command features, through practical examples
- Explore SQLite's basic language statements
- Connect to SQLite and execute SQL statements
- Extend the SQLite language to create your own software extensions
- Use HTML5 with PhoneGap on iOS
- Set up a Swift project using Xcode with SQLite
- Administer SQLite databases in an easy and effective way

Who this book is written for
This book is intended for those who want to learn about SQLite and how to develop apps in Swift or HTML5 using SQLite. Whether you are an expert Objective-C programmer or new to this platform, you'll learn quickly, grasping the code in real-world apps to use Swift.

What you will learn from this book
- Learn the background of SQL and SQLite for mobile development, its statements, and command features, through practical examples
- Explore SQLite's basic language statements
- Connect to SQLite and execute SQL statements
- Extend the SQLite language to create your own software extensions
- Use HTML5 with PhoneGap on iOS
- Set up a Swift project using Xcode with SQLite
- Administer SQLite databases in an easy and effective way

Prices do not include local sales tax or VAT

Gene Da Rocha, MSc, BSc (Hons)


Free Sample
In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 *Introduction to SQL and SQLite*
- A synopsis of the book’s content
- More information on *Learning SQLite for iOS*
About the Author

Gene Da Rocha, MSc, BSc (Hons) in mobile and computer science is an experienced IT professional with over 25 years in the IT industry. He has worked for a variety of companies nationally and internationally, in different industries including corporate, start-up, pharmaceutical, finance, banking, and the NHS.

Gene is also the owner and founder of a mobile solutions company, Voxstar (www.voxstar.com), based in London and Buckinghamshire. He comes from a programming and development background, and has worked with database technology, iOS, Android, Windows mobile, and a variety of other technologies.

He has been helping and advising, programming, and recently testing software for a number of companies such as DigitasLBI, Oxfam, News UK, QAWorks, Reuters, and the Association for Project Management, among many others.
SQLite is still a widely used database for mobile applications on smartphones and tablets. For those with SQL experience, it will be easier to understand and learn what it has to offer and the applications it can be used for. SQLite was released in 2000 has grown to be a well-used database for mobile device development.

Mr. D. Richard Hipp developed it on a battleship while he was at a company called General Dynamics. Initially used as storage, it was then developed using a B-tree implementation, which enhanced it and enabled the storage of rows and transactions.

This book gives you the opportunity to learn elements of SQLite, the mobile database; its interaction with the MAC operating system, Xcode; and the developer IDE for Apple apps and PhoneGap, which enables HTML5. It outlines how easy it is to work with SQLite.

**What this book covers**

*Chapter 1, Introduction to SQL and SQLite,* introduces you to the background of Structured Query Language (SQL) and the mobile database SQLite.

*Chapter 2, Database Design Concepts,* talks about the database concepts in SQLite.

*Chapter 3, Administering the Database,* introduces you to administering the SQLite database and makes you aware of the different components of this relational database.

*Chapter 4, Essentials of SQL,* this chapter talks about the essentials of SQL. It will outline the major possibilities with SQL and how it can be used properly on SQLite. This is essential so that you understand how SQL can be used and its limitations and advantages.

*Chapter 5, Exposing the C API,* deals with the C API and how you can extend its application use and produce the applications that you require using code.
Chapter 6, Using Swift with iOS and SQLite, looks at using the new programming language from Apple, Swift, with SQLite.

Chapter 7, iOS Development with PhoneGap and HTML5, looks at how to use Xcode with PhoneGap to integrate and compile with source code, including HTML5.

Chapter 8, More Features and Advances in SQLite, deals with how SQLite has changed in recent years, how it has advanced to be integrated into a variety of existing technologies, and how its simple easy-to-use formula has guaranteed its popularity with others.
1

Introduction to SQL and SQLite

In this chapter, I will introduce you to the Structured Query Language (SQL) and the mobile database SQLite. Whether you are an experienced technologist at SQL or a novice, using this book will help you understand this cool subject, which is gaining momentum. SQLite is a database that is used on a mobile smartphone or tablet, which is local to the device. SQLite has been modified by different vendors to harden and secure it for a variety of uses and applications.

SQLite was released in 2000 and has now grown to be de facto database on a mobile or smartphone. It is an open source piece of software with a low footprint and overheads, which is packed with a RDBMS (relational database management system).

Mr. D. Richard Hipp is the inventor and author of SQLite, which was designed and developed on a battleship while he was with a company called General Dynamics in the US Navy. The programming was built for the HP-UX operating system with Informix as the database engine. It took many hours in the data to upgrade or install the database software, and was an over-the-top database for this experienced DBA (database administrator). Mr. Hipp wanted a portable, self-contained, easy-to-use database, which could be mobile, quick to install, and not dependent on the operating system.

Initially, SQLite 1.0 used gdbm as its storage system, but later, it was replaced with its own B-tree implementation and technology for the database. The B-tree implementation was enhanced to support transactions and store rows of the data with key order. From 2001 onwards, open source family extensions for other languages, such as Java, Python, and Perl, were written to support their applications. The database and its popularity within the open source community and others started growing.
As described in Wikipedia, SQL was as follows:

Originally based upon relational algebra and tuple relational calculus, SQL consists of a data definition and manipulation language. The scope of SQL includes data insert, query, update and delete, schema creation and modification, and data access control. Although SQL is often described as, and to a great extent is, a declarative language (4GL), it also includes procedural elements.

Internationalization supported UTF-16 and UTF-8 and included text-collating sequences in versions 2 and 3 in 2004. It was funded by AOL (America Online) in 2004. It works with a variety of browsers that sometimes have in-built support for this technology. For example, there are many extensions that use Chrome or Firefox that allow you to manage the database.

There have been many features added to this product. The future with the growth in mobile phones sets this quick and easy relational database system to quantum leap, where this database's use within the mobile and tablet application space will increase.

SQLite is based on PostgreSQL as a point of reference. SQLite does not enforce any type checking. The schema does not constrain it since the type of value is dynamic, and a trigger will be activated by converting the datatype.

About SQL
In June 1970, a research paper was published by Dr. E.F. Codd called A Relational Model of Data for Large Shared Data Banks. The Association of Computer Machinery (ACM) accepted Codd's data and technology model, which has become the standard of the RDBMS today. IBM Corporation had invented the language called Structured English Query Language (SEQUEL), where the word "English" was dropped to become SQL.

SQL has become the standard for the RDMS, which is used by databases such as Oracle, Sybase, and Microsoft's SQL Server.

Today, there are American National Standards Institute (ANSI) standards for SQL, and there are many variations of this technology. Among the mentioned manufacturers, there are also others available in the open source world, for example, an SQL query engine, such as Presto.

Presto is the distribution engine for SQL under open source, which is made to execute interactive analytic queries. Presto queries are run under databases from a variety of data source sizes—gigabytes to petabytes.
Chapter 1

Companies such as Facebook and Dropbox use the Presto SQL engine for their queries and analytics in data warehouse and related applications.

SQL is made up of data manipulation and definition language built with tuple and algebra calculation in a relational format. This language has a variety of statements but most would recognize the INSERT, SELECT, UPDATE, and DELETE statements. These statements form a part of the database schema management process and aid the data and security accesses. SQL includes procedural elements as a part of its setup.

Where does SQLite stand in today's industry?

Companies may use applications, but they are not aware of the SQL engines that drive their data storage and information. Although it had become a standard with the ANSI in 1986, SQL features and functionalities are not 100% portable among different SQL systems. They also require code changes to be useful. These standards are always up for revision to ensure that ANSI is maintained.

There are many industrial and commercial databases, such as Oracle, SQL Server, or DB2, but none of them are as flexible, light, or open source as SQLite. Although smartphones are getting more powerful, you cannot compare them to the processing power of a modern desktop or laptop. SQLite, as its names suggests, is an SQL in a light environment, which is also flexible and versatile. So, at present, the best, light, fully functional, and customized database for mobile, is SQLite.

SQLite cannot be compared to enterprise database engines, such as SQL Server, Oracle, and MySQL. These enterprise database systems provide a centralized and controlled position, whereas SQLite provides local storage on a mobile device. SQLite is effectively based on the economy of size and reliability. It is simple to use, small, robust, and does not compete with these enterprise databases.

SQLite works well with "Internet of Things" as well, because of the no-need-for-human input or administration feature. So, for applications that deal with drones, medical equipment, robots, and sensors, SQL makes an ideal candidate for usage on a variety of mobile applications.
iOS with SQLite

Out of the hundreds of thousands of apps in all the app stores, it would be difficult to find the one that does not require a database of some sort to store or handle data in a particular way. There are different formats of data and datafeeds, but they all require some sort of temporary or permanent storage. A small amount of data may not be applicable, but a medium or large amount of data will require a storage mechanism, such as a database to assist the app.

Using a database such as SQLite with iOS will enable developers to use their existing skills to run their DBMS. For SQLite, there is a C-library that is embedded and available to use with iOS with the Xcode IDE.

Apple fully supports SQLite, which uses an include statement as a part of the library call, but developers can also use FMDB, which is a cocoa/objective-C wrapper around SQLite.

A few advantages of SQLite are that it is fast, lightweight, reliable, uses existing SQL knowledge, is supported by Apple on Mac OS and iOS and by many developers, as well as being integrated without much outside involvement.

The SQLite 3 library is under the general tab once the main project name is highlighted on the left-hand side of the page. Then, at the bottom of the page, within Linked Frameworks and Libraries, click on the + for a modal window to appear. Enter the word sqlite and select the libsqlite3.dylib library, as shown in the following screenshot:
In effect, it is the C++ wrapper, called the `libsqlite3.dylib` library, within the framework section that allows the API to work with SQLite commands.

Before any SQL processes can take place, the database should be opened and ready for querying, and, upon the success of data retrieval, the constant called `SQLITE_OK` should be set to 0.

Once the C++ wrapper is used and the access to SQLite commands is available, it is an easier process to use SQLite with iOS.
Embedded databases

SQLite has been designed and developed to work and coexist with other applications and processes in its area. RDBMS is tightly integrated with the native application software that requires storing information but is masked, which is hidden from users, and it requires minimal administration or maintenance.

SQLite can work with different APIs hidden from users, and it requires minimal administration or maintenance areas.

The RDMS SQLite will also work with other applications:

- It requires minimal supervision
- There is no network traffic and access is faster since it is a file-based system
- There are no network access conflicts or configurations
- There are no access limitations with privileges or permissions
- There is much reduced overheads

These make it easier and quicker to deploy your applications to app stores or other locations.

*Figure 1*, seen in this section, shows how different components work seamlessly together in a harmonized way to link up data with the SQLite library and other processes. These show how the Apache and C/C++ processes work together with the SQLite-C library to interface and link with it, so it becomes seamless and integrates with the operating system.

SQLite has been developed and integrated in such a way that it will interface and gel with a variety of applications and multiple solutions. As a lightweight RDBMS, it can stand on its own due to its versatility and is not cumbersome or too complex to benefit your application. It can be used on many platforms, and comes with a binary compatible format, which is easier to dovetail within your mobile application.

The different types of IT professionals will be involved with SQLite, since it holds the data, affects performance, and involves database design, user or mobile interface design specialists, analysts, and consultancy types. These professionals can use their existing knowledge of SQL to quickly grasp SQLite. SQLite can act as both data processor for information, or deal with data in the memory, to perform in an excellent manner.

*Figure 1* also outlines how the different software pieces of a jigsaw can interface properly using the C API interface for SQLite with some other programming language code. For example, C or C++ code can be programmed to communicate with the SQLITE C API, which will then talk to the operating system and communicate with the database engine. Another language, such as PHP, can communicate using its own language data objects, which will, in turn, communicate with the SQLite C API and the database.

SQLite is a great database to learn, especially for computer scientists who want to use a tool that can open their minds to investigate caching, B-Tree structures and algorithms, database design architecture, and other concepts.
Introduction to SQL and SQLite

For more information of how SQLite sits within the other applications on a mobile device, see Figure 1:

![Diagram of C API interface to SQLite](image)

Figure 1: C API interface to SQLite

The architecture of the SQLite database

As a library within the OS-Interface, SQLite will have many functions implemented through a program called tclsqlite.c. Many technologies and reserved words are used in different languages, but here we have used C language. The core functions are to be found in main.c, legacy.c, and vmbeapi.c. There is also a source code file in C for the TCL language, to avoid any confusion; the prefix of sqlite3 is used at the beginning of the SQLite library.

The **Tokenizer** code base is found within tokenize.c. Its task is to look at the strings that are passed to it and partition or separate them into tokens, which are then passed to the parser. The tokenize.c file is included in the code with an include statement and is located in the sqlite/src/tokenize.c directory area.

The **Parser** code base is found within parse.y. The **Lemon LALR(1)** parser generator is the parser for SQLite; it takes the concept of tokens and assigns them a meaning. To keep within the low-sized footprint of RDBMS, only one C file is used for the parse generator.

The **Code Generator** is then used to create SQL statements from the outputted tokens of the parser. It will produce some virtual machine code that will carry out the work of SQL statements. Several files, such as attach.c, build.c, delete.c, select.c, and update.c, will handle the SQL statements and syntax.
Virtual machines execute the code that is generated from the Code Generator. It has in-built storage, where each instruction may have up to three additional operands as a part of each code. The source file is called vdbe.c, which is a part of the SQLite database library. Built in is also a computing engine that has been specially created to integrate with the database system.

There are two header files for virtual machines. The header files that interface a link between the SQLite libraries are vdbe.h and vdbeaux.c, which have utilities used by other modules. The vdbeapi.c file also connects to virtual machines with sqlite3_bind and other related interfaces. C language routines are called from SQL functions to reference them to the header files. For example, functions such as count() are defined in func.c, and date functions are located in date.c.

B-tree is a type of table implementation used in SQLite, and the C source file is btree.c. The btree.h header file defines the interface of the B-tree system. There is a different B-tree setup for every table and index held within the same file. There is a header portion within btree.c, which will have details of B-tree in a large comment field.

Pager or Page Cache using B-tree will ask for data in a fixed size format. The default size is 1024 bytes, but it can be between 512 and 65536 bytes. Commit and Rollback operations, coupled with the caching, reading, and writing of the data, are handled by Page Cache or Pager. Data locking mechanisms are also handled by Page Cache. The C file called page.c is implemented to handle requests within the SQLite library and the header file is pager.h.

The OS Interface C file is defined in os.h. It addresses how SQLite can be used on different operating systems, and it becomes transparent and portable to the user, thus becoming a valuable solution for any developer. An abstract layer to handle Win32 and POSIX compliant systems is also kept in place. Different operating systems have their own C file. For example, os_win.c is for Windows, os_unix.c is for Unix; both are coupled with their own os_win.h and os_unix.h header files.

Util.c is the C file that will handle memory allocation and string comparisons. The Utf.c C file will hold Unicode conversion subroutines.
Introduction to SQL and SQLite

For more information on the architecture of SQLite, see Figure 2:

![Architecture diagram of SQLite](image)

Figure 2: Architecture diagram of SQLite

The utf.c file will hold the Unicode data, sort it within the SQL engine, and use the engine as a mechanism for computing data. Since the memory of the device is limited and the database size has the same constraints, the developer has to think outside the box to use these techniques.

These types of memory and resource management formed a part of the approach when the overlay techniques were used in the past and the disk and memory was limited:

```sql
SELECT parameter1, STTDEV(parameter2)
FROM Table1 Group by parameter1
HAVING parameter1 > MAX(parameter3)
```

### Features

As part of its standards, SQLite uses and implements most of the SQL-92 standards, but not all the potential features or parts of functionality are used or realized. For example, SQLite uses and implements most of the SQL-92 standards but not all potent columns. The support for triggers is not 100% as it cannot write output to the views.
As mentioned previously, the use of a common datatype for a column is different; most relational database systems assign them to individual values. SQLite will convert a string to an integer if the column's preferred type is an integer. It is a good piece of functionality when bound to this type of scripting language, but the technique is not portable to other RDBMS systems. It also has its criticisms for not having a good data integrity mechanism compared to others, in relation to statically typed columns.

There are some major differences between the two approaches of data: one is Core Data and the SQLite way Core Data is similar to having a layer of information between the user interface and the database itself. It does have the advantage of speeding up database interactivity read/write process and saves writing huge queries. While SQLite is a local relational database with its own efficiencies and limitations, it may not suit all applications. Sometimes, due to the speed of a device, Core Data may have the advantage of SQLite directly.

Briefly, your application will have model layer objects, and these are held and managed in a framework, namely, Core Data. It manages the life cycle of an object within iOS. This is just some background information to show how data can be read from databases and the speed at which this takes place.

The advantages of using SQLite

A few advantages of using SQLite are listed here:

- SQLite does have a data constraints feature and can edit or drop tables without loading them into memory.
- SQLite works on the data stored on the disk and is slower compared to Core Data.
- Core Data, on the other hand, does not have data constraints, and can be implemented using the app's business logic instead.
- In order to update or drop a table, the entire table has to be loaded up.
- Core Data is quick to create records/rows but slower to save the data.
- Core Data does have another advantage where it operates and works in the memory, and the data has to be loaded from the disk to memory.
- Core Data works with objects based in the memory, or can be accessed using the standard slower disk method.
- Core Data will work on non-transactional, single user, or single-threaded methods. SQLite's function is to fetch and store data using its file database system. It operates by storing the data on the disk where the data is incrementally or minimally loaded.
Effectively, the data can be transactional, multiuse, and thread-safe. It saves data to the disk and is mostly resilient to crashes. It is slower if you have to create hundreds of thousands of rows, but it does have data constraints, such as unique keys.

SQLite has bindings to many languages such as Basic, C, C#, C++, Java, JavaScript, Lua, PHP, Objective-C, Python, Ruby, and TCL. Its popularity with the open source community and usage by customers and developers has enabled its growth to continue.

This lightweight RDMS can be used on Google Chrome, Firefox, Safari, Opera, and Android browsers and has middleware support using ADO.NET, ODBC, COM (ActiveX), and XULRunner. It also has a support for web application frameworks, such as Django (Python based), Ruby on Rails, and Bugzilla (Mozilla). There are other applications, such as Adobe Photoshop Light and Skype that use SQLite. It is also a part of Windows 8, Symbian OS, Android, and OpenBSD operating systems.

Apart from not having the large overheads of other database engines, SQLite has a major enhancement, known as the EXPLAIN keyword, with its manifest typing.

For controlling constraint conflicts, the REPLACE and ON CONFLICT statements are used.

Within the same query, multiple independent databases can be accessed using the DETACH and ATTACH statements.

New SQL functions and collating sequences can be created using the predefined APIs, which offer much more flexibility.

As there is no configuration required, SQLite just does the job and works.

No lists, such as the REPLACE and ON CONFLICT procedures, are required.

There is no need to initialize, stop, restart, or start server processes, and no administrator is required to create the database with proper access controls or security permits.

After any failure, no user actions are required to recover the database, since it is self-repairing.

SQLite is more advanced than previously thought. Unlike other RDMS, it does not require a server setup via a server to serve up data or incur network traffic costs. There is no TCP/IP calls, nor frequent communication backward or forward.

SQLite is direct; the operating system process deals with database access to its file and controls database writes and reads with no middle-man process handshaking.
• By having no server backend, the process of installation, configuration, or 
administration is reduced significantly, and access to the database is granted 
to programs that require this type of data operation. This is an advantage in 
one way, but it is also a disadvantage for security and protection from data-
driven misuse, data concurrency, or data row locking mechanisms.
• It also allows the database to be accessed several times by different 
applications at the same time.
• It supports a form of portability for the cross platform database file that can 
be located with the database file structure. The database file can be updated 
on one system and copied to another on either 32 bit or 64 bit with different 
arichitectures; this does not make a difference to SQLite.
• The usage of different architectures and the promises of developers to keep 
the file system stable and compatible with previous, current, and future 
developments will allow this database to grow and thrive. SQLite databases 
do not need to upload old data to new, formatted, and upgraded databases; 
it just works.
• By having a single disk file for the database, the information can be copied on 
a USB and shared or just reused on another device very quickly by keeping 
all the information intact.
• Another feature of this portable database, SQLite, is its size, which can 
start on a single 512-byte page and expand to 2,147,483,646 pages at 65,536 
bytes per page, or in bytes 140,737,488,224,256, which equates to about 140 
terabytes. Most other RDBMS are much larger, but IBM's Cloudscape is 
small, with a 2 MB jar file, but still larger than SQLite.
• The Firebird alternative's client (frontend) library is about 350 KB, whereas 
the Berkeley Oracle database is around 450 KB, without the SQL support, 
and with one simple key/value pair's option.
• This advanced portable database system and its source code is in the public 
domain. However, there are open source license issues and controls for some 
test code and documentation.
• This is great news for developers who might want to code up new extensions 
or database functionality that works with their programs, which could be 
made into a "product extension" for SQLite.
• You cannot have this sort of access to SQL source code around since 
everything has a patent, limited access, or just no access.
• There are signed affidavits by developers to disown any copyright interest in 
the SQLite code. SQLite is different because it is just not governed or ruled 
by copyright law, which monitors the way a software should really work or 
be used.
Using the small allocation with variable length records, applications run faster, database access is quicker, manifest typing is used, and the database is small and nimble.

The ease of using this RDBMS makes it easier for most programmers at an intermediate level to create applications using this technology, with its detailed documentation and examples.

Other RDBMS are internally complex, with links to data structures and objects. SQLite comprises a virtual machine language that uses the `EXPLAIN` reserved word in front of a query.

The virtual machine has increased and benefitted this database engine by providing an excellent process or controlled environment between the backend (where the results are computed and outputted), and frontend (where the SQL is parsed and executed).

The SQL implementation language is comparable to other RDBMS, especially with its lightweight base, and it supports recursive triggers and requires the `FOR EACH ROW` behavior. The `FOR EACH` statement is not currently supported, but functionality cannot be ruled out in the future.

As described so far in this chapter, SQLite is a nimble and easy-to-use database that developers can engage with quickly, use existing skills, and output systems to mobile devices and tablets far easier than ever before. With the help of HTML5 and other JavaScript frameworks, the advancement of SQL and number of SQLite installations will take a quantum leap.

**Working with SQLite**

The website for SQLite is available at [www.sqlite.org](http://www.sqlite.org), where you can download all the binaries for the database, documentation, and source code, which works on operating systems such as Linux, Windows, and MAC OS X.

The SQLite share library or DLL is the library to be used for the Windows operating system and can be installed or seen via Visual Studio with the C++ language. So, the developer can write the code using the library that is presently linked in reference via the application. When the execution has taken place, the DLL will load and all the references in the code will link to those in the DLL at the right time.

The SQLite3 command-line program—CLP—is a self-contained program that has all the components to run at the command line.

It also comes with an extension for TCL. So within TCL, you can connect and update the SQLite database. SQLite downloads come with the TAR version for Unix systems, and the ZIP version for Windows systems.
The examples of using SQLite with iOS

The following is a simple application on how to use iOS with the SQLite database with Xcode. It outlines the basic steps of creating an application and database, and selecting data.

To get started, let's start Xcode and create a template using the Single View Application choice, as shown in the following screenshot:
Click on the **Next** button to proceed to the next screen in this process. In the product name field, enter **SimpleCalculator** for the language, and select **Swift**. For the devices field, select **iPhone**. Then, click on the **Next** button to move onto the next screen, as shown in the following screenshot:

In the following screenshot, select the directory where the code will reside. Now, we can view what the Xcode developer tool has created. Then, select a device to display the information; in our case, use the iPhone 6s.
See the directory for the source code, as shown in the following screenshot:

The following is a screenshot showing the SimpleCalculator application opened in Xcode. Select the iPhone 6s option as the device to develop on:
Next, click on the Play button that will compile and build the application as shown here in both images, and a blank screen will appear:

The preceding screenshot shows the application to be built, and the following screenshot shows a blank screen after the image is compiled and run.

This method gets you to the basics of an iOS application with Swift working as a canvas.

In this brief example, we will use SQLiteDB.swift and String-Extras.swift to work with the SQLite database, including the Bridging-Header.h file. In the Build Settings option, view Objective-C Bridging Header and double-click on it, and bridge it to Bridging-Header.h, and you can also drag it to show that it is linked.

As mentioned previously, add libsqlite3.0.dylib to the linked frameworks by navigating to General | Linked Frameworks and Libraries; then, add Libsqlite3.0.dylib.

Now, rebuild the project to show that it's working:
Click on the **Simulator** button, and then click on **Quit** to stop the current compiled simulator program. The program will compile with no problems. Next, a database instance has to be created as shown in the following code. The `SQLite.DB.query` method is used to execute these commands:

- First an instance is required:
  
  ```swift
  let testdb = SQLiteDB.sharedInstance()
  ```

- To run this query, the following code is used with the `SQLiteDB.query` way:
  
  ```swift
  var theresult = testdb.query("select * from people where county = 'Berks'", parameters: nil)
  for row in result
  {
      println(row["name"]!.asString())
  }
  ```

- To delete a record for example, follow the following piece of code:
  
  ```swift
  testdb.execute("delete from people where county = 'Bucks'", parameters: nil)
  ```

**Summary**

In this chapter, you read the history of SQL, the impact of relational databases, and the use of a mobile SQL database, namely, SQLite. This chapter outlined the history and beginnings of SQLite and how it has grown to be the most used database on mobile devices so far. In the next chapter, you will learn about the components of database concepts and how to design an SQLite database. The next chapter will show you the basic elements of design for an SQLite database.
Where to buy this book
You can buy Learning SQLite for iOS from the Packt Publishing website.
Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.
Click here for ordering and shipping details.