

# B4X Booklets

B4A

B4i

B4J

## B4X SQLite Database

1	B4X platforms.....	5
2	SQLite Database.....	6
2.1	General information .....	6
2.2	SQLite Database basics.....	8
2.2.1	Database initialization SQL1.Initialize / SQL1.InitializeSQLite.....	8
2.2.2	Table creation CREATE TABLE .....	9
2.2.3	INTEGER PRIMARY KEY rowid.....	10
2.2.4	Adding data INSERT INTO.....	11
2.2.5	Updating data UPDATE .....	11
2.2.6	Reading data SELECT .....	11
2.2.7	Filtering WHERE.....	13
2.2.8	Sorting ORDER BY .....	14
2.2.9	Date / Time functions.....	15
2.2.10	Other functions.....	17
2.2.10.1	Get the data type of columns typeof().....	17
2.2.10.2	Get the max length of the data in a column length().....	17
2.2.10.3	Get a sub string substr() .....	17
2.2.10.4	Replace parts of a string replace().....	18
2.2.10.5	Find a substring in a string instr() .....	18
2.2.10.6	Round a number round() .....	18
2.2.10.7	Get the total number of rows count() .....	18
2.2.10.8	Get the tables in the database sqlite_master.....	18
2.2.10.9	Get the column names of a table TableName .....	18
2.2.10.10	Get the number of database rows that were changed changes() .....	19
2.2.10.11	Get the PRIMARY KEYs from a table rowid .....	19
2.2.11	ResultSet GetInt, GetInt2 etc B4A, B4i, B4J.....	20
2.2.12	Cursor GetInt, GetInt2 etc B4A only .....	21
2.2.13	Get Table information PRAGMA.....	22
2.2.14	Deleting data DELETE FROM.....	23
2.2.15	Rename a table ALTER TABLE Name RENAME TO.....	23
2.2.16	Add a column ALTER TABLE Name ADD COLUMN.....	23
2.2.16.1	Update the database after having added a column .....	23
2.2.17	Delete a table DROP TABLE .....	23
2.2.18	Insert an image .....	24
2.2.19	Read an image .....	24
2.2.20	ExecQuery vs ExecQuery2 / ExecNonQuery vs ExecNonQuery2.....	25
2.2.21	Insert many rows SQL.BeginTransaction / SQL.EndTransaction .....	26
2.2.22	Asynchronous queries .....	27
2.2.23	Batch inserts AddNonQueryToBatch / ExecNonQueryBatch .....	28
2.3	Multiple tables.....	29
2.4	Transaction speed.....	31
2.5	First steps .....	32
2.5.1	Reference the SQLite library .....	32
2.5.2	Declare the SQLite library .....	32
2.5.3	Initialize the SQLite library and the variables .....	33
2.6	SQLite Database first simple example program SQLiteLight1 .....	36
2.6.1	Source code .....	37
2.6.1.1	B4A Program initialization .....	37
2.6.1.2	B4i Program initialisation .....	39
2.6.1.3	B4J Program initialisation.....	40
2.6.2	Database handling .....	41
2.6.2.1	Create database.....	41
2.6.2.2	ReadDataBase .....	41

2.6.2.3	ShowEntry .....	42
2.6.2.4	AddEntry .....	43
2.6.2.5	DeleteEntry .....	44
2.6.2.6	UpdateEntry .....	44
2.7	SQLite Database second simple example program SQLiteLight2 .....	45
2.7.1	Main module source code parts.....	47
2.7.1.1	Declaration of the Process global variables .....	47
2.7.1.2	Show table.....	48
2.7.1.3	ExecuteHtml show a table in a WebView.....	49
2.7.1.4	ReadDatabaseRowIDs.....	50
2.7.1.5	UpdateSelectedEntryDisplay .....	50
2.7.1.6	WebView events _ OverrideUrl / _LocationChanged .....	51
2.7.2	Edit Module source code parts .....	52
2.7.3	Filter Module source code parts .....	52
2.7.3.1	B4A .....	52
2.7.3.2	B4i .....	53
2.7.3.3	B4J .....	53
2.8	SQLite Database third simple example program SQLiteLight3 .....	54
2.9	SQLite Database 3 <sup>rd</sup> example with B4XTable .....	55
2.10	SQLite Database 3 <sup>rd</sup> example XUI version SQLiteLight3X.....	56
2.11	SQLite Database fourth example program SQLiteLight4 .....	57
2.12	SQLite Viewer .....	59
3	DBUtils version 2.....	60
3.1	DBUtil functions .....	61
3.1.1	CopyDBFormAssets B4A, B4 .....	62
3.1.2	CopyDBFormAssets B4J .....	62
3.1.3	CreateTable B4A, B4i, B4J.....	62
3.1.4	DeleteRecord B4A, B4i, B4J .....	62
3.1.5	DropTable B4A, B4i, B4J .....	63
3.1.6	ExecuteHtml B4A, B4i, B4J .....	63
3.1.7	ExecuteJSON B4A, B4i, B4J .....	63
3.1.8	ExecuteList B4A, B4i, B4J .....	63
3.1.9	ExecuteListView B4A .....	64
3.1.10	ExecuteMap B4A, B4i, B4J .....	64
3.1.11	ExecuteMemoryTable B4A, B4i, B4J .....	64
3.1.12	ExecuteTableView B4J .....	64
3.1.13	ExecuteSpinner B4A .....	64
3.1.14	GetDBFolder B4A, B4i, B4J .....	65
3.1.15	GetDBVersion B4A, B4i, B4J .....	65
3.1.16	GetFieldInfo B4A, B4i, B4J .....	65
3.1.17	GetTables B4A, B4i, B4J .....	65
3.1.18	InsertMaps B4A, B4i, B4J .....	65
3.1.19	SetDBVersion B4A, B4i, B4J .....	65
3.1.20	TableExists B4A, B4i, B4J .....	65
3.1.21	UpdateRecord B4A, B4i, B4J .....	66
3.1.22	UpdateRecord2 B4A, B4i, B4J .....	66
3.2	Examples .....	66
3.3	DBUtilsDemo example program .....	67
3.3.1	Code differences.....	69

Main contributors: Klaus Christl (klaus), Erel Uziel (Erel)

**To search for a given word or sentence use the Search function in the Edit menu.**

All the source code and files needed (layouts, images etc.) of the example projects in this guide are included in the SQLiteDatabase\_SourceCode folder.

Updated for following versions:

B4A version 11.0

B4i version 7.50

B4J version 9.10

[B4X Booklets:](#)

B4X Getting Started

B4X Basic Language

B4X IDE Integrated Development Environment

B4X Visual Designer

B4X Help tools

B4XPages Cross-platform projects

B4X CustomViews

B4X Graphics

B4X XUI **B4X User Interface**

B4X SQLite Database

B4X JavaObject NativeObject

B4R Example Projects

You can consult these booklets online in this link [\[B4X\] Documentation Booklets](#).

Be aware that external links don't work in the online display.

## 1 B4X platforms

B4X is a suite of BASIC programming languages for different platforms.

B4X suite supports more platforms than any other tool

ANDROID | IOS | WINDOWS | MAC | LINUX | ARDUINO | RASPBERRY PI | ESP8266 | AND MORE...

- **B4A**  **Android**

B4A is a **100% free** development tool for Android applications, it includes all the features needed to quickly develop any type of Android app.

- **B4i**  **iOS**

B4i is a development tool for native iOS applications.

B4i follows the same concepts as B4A, allowing you to reuse most of the code and build apps for both Android and iOS.

- **B4J**  **Java / Windows / Mac / Linux / Raspberry PI**

B4J is a **100% free** development tool for desktop, server and IoT solutions.

With B4J you can easily create desktop applications (UI), console programs (non-UI) and server solutions.

The compiled apps can run on Windows, Mac, Linux and ARM boards (such as Raspberry Pi).

- **B4R**  **ARDUINO** **Arduino / ESP8266**

B4R is a **100% free** development tool for native Arduino and ESP8266 programs.

B4R follows the same concepts of the other B4X tools, providing a simple and powerful development tool.

B4R, B4A, B4J and B4i together make the best development solution for the Internet of Things (IoT).

- **B4XPages**

B4XPages is an internal library for B4A, B4i and B4J allowing to develop easily cross-platform programs.

B4XPages is explained in detail in the B4XPages Cross-platform projects booklet.

Even, if you want to develop only in one platform it is interesting to use the B4XPages library it makes the program flow simpler especially for B4A.

## 2 SQLite Database

### 2.1 General information

This guide covers the use of SQLite Databases in the B4X languages (B4A, B4i, B4J).

All the source code and files needed (layouts, images etc) of the example projects in this guide are included in the `SQLiteDatabase_SourceCode` folder.

There are three folders for each project, one for each platform B4A, B4i and B4J.

What is a database (source Wikipedia [Database](#)):

A **database** is an organized collection of data for one or more purposes, usually in digital form. The data are typically organized to model relevant aspects of reality (for example, the availability of rooms in hotels), in a way that supports processes requiring this information (for example, finding a hotel with vacancies). The term "database" refers both to the way its users view it, and to the logical and physical materialization of its data, content, in files, computer memory, and computer data storage. This definition is very general, and is independent of the technology used. However, not every collection of data is a database; the term database implies that the data is managed to some level of quality (measured in terms of accuracy, availability, usability, and resilience) and this in turn often implies the use of a general-purpose [Database management system](#) (DBMS). A general-purpose DBMS is typically a complex software system that meets many usage requirements, and the databases that it maintains are often large and complex.

The interface between your program and the database is the SQL language.

The data is stored in tables, each table has a certain number of columns and rows.

Each row contains a data set and the different data of a given set are stored in the columns.

Simple example programs are included in the `SourceCode\SQL` folder.

**If you add a default database to your project in the files Tab, it is in the DirAssets folder.**

**Databases cannot be accessed in DirAssets even if it is only for reading.**

Therefore, you must copy it to another folder.

Example in the SQLiteLight1 programs.

#### B4A

With DirInternal.

For example: DirInternal

Example code in the Starter module:

```
If File.Exists(File.DirInternal, "Database.db") = False Then
    File.Copy(File.DirAssets, "Database", File.DirInternal, "Database.db")
End If
SQL1.Initialize(File.DirInternal, "Database.db", True)
```

Or in Activity\_Create if you have only one Activity:

```
If FirstTime Then
    If File.Exists(File.DirInternal, "Database.db") = False Then
        File.Copy(File.DirAssets, "Database", File.DirInternal, "Database.db")
    End If
    SQL1.Initialize(File.DirInternal, "Database.db", True)
End If
```

#### B4i

In **Application\_Start**. We use the default directory for the application.

```
'check if the database already exists
If File.Exists(File.DirDocuments, "persons.db") = False Then
    'copy the default DB
    File.Copy(File.DirAssets, "persons.db", File.DirDocuments, "persons.db")
End If
SQL1.Initialize(File.DirDocuments, "persons.db", True)
```

#### B4J

In **App\_Start**. We use the default directory for the application.

```
'check if the database already exists
If File.Exists(File.DirData("jSQLiteLight2"), "persons.db") = False Then
    'copy the default DB
    File.Copy(File.DirAssets, "persons.db", File.DirData("jSQLiteLight2"), "persons.db")
End If
SQL1.InitializeSQLite(File.DirData("jSQLiteLight2"), "persons.db", True)
```

## 2.2 SQLite Database basics

Some simple SQL instructions.

Here you find the SQLite site: [SQLite](#)

Here you find the SQLite syntax: [SQLite syntax](#)

A very interesting website to learn SQL is this one: [W3Schools SQL](#).

And another one: [SQLiteTutorial](#).

### 2.2.1 Database initialization SQL1.Initialize / SQL1.InitializeSQLite

To use a database, you must first initialize it!

This is ideally done in the **Starter** service for B4A, in **Application\_Start** for B4i and in **App\_Start** for B4J.

#### B4A, B4i

```
SQL1.Initialize(DBDirName, DBFileName, True)
```

DBDirName = Directory name of the database.

DBFileName = Database file name.

True = Create if necessary False don't create the database.

```
SQL1.Initialize(DBDirName, DBFileName, True)
```

#### B4J

Add the line below in Main module #Region Project Attributes.

```
#AdditionalJar: sqlite-jdbc-3.7.2
```

And:

```
SQL1.InitializeSQLite(DBDirName, DBFileName, True)
```

DBDirName = Directory name of the database.

DBFileName = Database file name.

True = Create if necessary False don't create the database.

```
SQL1.InitializeSQLite(DBDirName, DBFileName, True)
```

In B4J, SQL1.Initialize is used to initialize SQL drivers.

To use SQLite, you must initialize it with SQL1.InitializeSQLite.

#### B4A, B4i, B4J

If you want to use a database only in the current instance of the program without saving somewhere else, you can initialize it like this:

```
#If B4J
```

```
SQL1.InitializeSQLite("", ":memory:", True)
```

```
#Else If B4A OR B4i
```

```
sql1.Initialize("", ":memory:", True)
```

```
#End If
```



### 2.2.2 Table creation CREATE TABLE

You can create a database in a SQLite program on the PC or you can create it in the code like below.

```
CREATE TABLE TableName (Col1 INTEGER, Col1 TEXT, Col2 REAL )
```

Creates a table with the name 'TableName' and three columns:

Column Index	Name	Variable Type
1	Col1	INTEGER
2	Col2	TEXT
3	Col3	REAL

```
SQL1.ExecNonQuery("CREATE TABLE TableName(Col1 INTEGER, Col2 TEXT, Col3 REAL")
```

Only these data types are available:

INTEGER is a 64-bit signed integer number.

REAL is a 64-bit IEEE floating point number.

TEXT is a string.

BLOB **B**inary **L**arge **O**bject, the value is stored exactly as it was input.

NULL

INTEGER PRIMARY KEY is a special variable type used for identifier ID's. It is a long integer value beginning with 1 and it is incremented by one each time a new data set is added to the database.

SQL identifiers are case insensitive. You could use for example:

```
SQL1.ExecNonQuery("CREATE TABLE TableName(col1 integer, col2 text, col3 Real")
```

But in B4A, SQL.GetString(ColumnNane), ColumnNane is case sensitive!

The column names must be spelled exactly the same name as in the table creation.

With the example above, col1 works but Col1 will throw an error.

### 2.2.3 INTEGER PRIMARY KEY      rowid

INTEGER PRIMARY KEY is a special data type which is unique and will never change. You can define a specific column dedicated to the PRIMARY KEY.

But this is not mandatory, SQLite has an internal column named *rowid* which can be used.

This is used in the SQLiteLight examples.

Each time you add a new record the PRIMARY KEY is incremented by 1.

When you delete a record the PRIMARY KEY of this record is **lost**.

When you load a database and display it in a table be aware that the row indexes in the table are not the same as the database rowids. Therefore, you must read and memorize the PRIMARY KEYs somewhere to know which record is in which line.

Comparison:

- Creation.
  - With a specific ID column.  
`"CREATE TABLE persons (ID INTEGER PRIMARY KEY, FirstName TEXT, LastName TEXT, City TEXT)"`
  - With no specific ID column.  
`"CREATE TABLE persons (FirstName TEXT, LastName TEXT, City TEXT)"`
- Reading.
  - With a specific ID column.  
`"SELECT ID, FirstName AS [First name], LastName AS [Last name], City FROM persons"`
  - With no specific ID column.  
 Reads the PRIMARY Key in the query.  
`"SELECT rowid AS ID, FirstName AS [First name], LastName AS [Last name], City FROM persons"`  
 Doesn't read the PRIMARY Key in the query.  
`"SELECT FirstName AS [First name], LastName AS [Last name], City FROM persons"`

Note: If you use this query `"SELECT * FROM persons"` the rowid column is not included. If you want it, you must specify it like in the examples above.  
 Read it like this `"SELECT rowid, * FROM persons"` or read it in a separate query.

You can use alias column names with the **As** keyword like:  
`"SELECT LastName AS Name, City FROM persons"`  
 or  
`"SELECT FirstName AS [First name], LastName AS [Last name] FROM persons"`  
 If there are spaces in the text you need to add square brackets like in the line above.
- Inserting.
  - With a specific ID column.  
`"INSERT INTO persons VALUES (NULL, 'John', 'KERRY', 'Boston')"`  
 You must use **NULL** for the PRIMARY KEY column.
  - With no specific ID column.  
`"INSERT INTO persons VALUES ('John', 'KERRY', 'Boston')"`

### 2.2.4 Adding data **INSERT INTO**

`INSERT INTO TableName VALUES ( Val1, Val2, Val3 )`

`SQL1.ExecNonQuery("INSERT INTO TableName VALUES (Val1, Val2, Val2)")`

If you enter values like this:

`SQL1.ExecNonQuery("INSERT INTO TableName VALUES (12, 'John', 235)")`

In this case, texts must be between two quotes like 'John', numbers not like 12 and 235.

Or

`SQL1.ExecNonQuery2("INSERT INTO TableName VALUES (?, ?, ?)" Array As String(Val1, Val2, Val2))`

`SQL1.ExecNonQuery2("INSERT INTO TableName VALUES (?, ?, ?)" Array As String(12, "John", 235))`

### 2.2.5 Updating data **UPDATE**

`UPDATE TableName Set Col1 = Val1, Col2 = 'Val2', Col3 = Val3 WHERE ID = idVal`

`SQL1.ExecNonQuery("UPDATE TableName Set Col1 = Val1, Col2 = 'Val2', Col3 = Val3 WHERE ID = idVal")`

Again, a text variable must be between two quotes like 'Val2', numbers not like Val1 and Val3.

Or

`SQL1.ExecNonQuery2("UPDATE TableName Set Col1 = ?, Col2 = ?, Col3 = ? WHERE ID = ?" Array As String(Val1, Val2, Val3, idVal))`

Here no need to care with the quotes for text variables!

### 2.2.6 Reading data **SELECT**

The SELECT statement is used to query the database. The result of a SELECT is zero or more rows of data where each row has a fixed number of columns.

A SELECT statement does not make any changes to the database.

Examples:

- The entire database:  
`SELECT * FROM TableName`  
`ResultSet1 = SQL1.ExecQuery("SELECT * FROM TableName")`
- A single column  
`SELECT Col1 FROM TableName`  
`ResultSet1 = SQL1.ExecQuery("SELECT Col1 FROM TableName")`
- Distinct values from a column, no duplicate values  
`SELECT DISTINCT Col1 FROM TableName`  
`ResultSet1 = SQL1.ExecQuery("SELECT DISTINCT Col1 FROM TableName")`
- Single entry (value)  
`SELECT Col1 FROM TableName WHERE rowid = idVal`  
`Value = SQL1.ExecQuerySingleResult("SELECT Col1 FROM TableName WHERE rowid = idVal")`

- Max / min value in a column, in the examples the max and min values of the given column.

```
SELECT max(Col1) FROM TableName
```

```
SELECT min(Col1) FROM TableName
```

```
Max = SQL1.ExecQuerySingleResult("SELECT max(Col1) FROM TableName")
```

```
Min = SQL1.ExecQuerySingleResult("SELECT min(Col1) FROM TableName")
```

- Get the sum or average of a column

```
SELECT total(Col1) FROM TableName
```

```
SELECT avg(Col1) FROM TableName
```

```
Sum = SQL1.ExecQuerySingleResult("SELECT total(Col1) FROM TableName")
```

```
Average = SQL1.ExecQuerySingleResult("SELECT avg(Col1) FROM TableName")
```

There exist also a sum() function, but it's better to use total().

If there is a row with a Null value, sum() returns Null, but Total() returns 0!

- Get calculations of columns.

For example, in a database with a column *Number* of type INTEGER and another column *Price* of type REAL we want to get the Cost = Number \* Price.

```
SELECT Number, Price, Number * Price FROM TableName
```

```
ResultSet1 = SQL1.ExecQuery("SELECT Number, Price, Number * Price FROM  
TableName")
```

```
Number = ResultSet1.GetInt2(0)
```

```
Price = ResultSet1.GetDouble2(1)
```

```
Cost = ResultSet1.GetDouble2(2)
```

Or giving the result a column name with Number \* Price AS Cost.

```
ResultSet1 = SQL1.ExecQuery("SELECT Number, Price, Number * Price AS Cost FROM  
TableName")
```

```
Number = ResultSet1.GetInt("Number")
```

```
Price = ResultSet1.GetDouble("Price")
```

```
Cost = ResultSet1.GetDouble("Cost")
```

Some functions:

- sum()            Calculates the sum of a column.
- avg()            Calculates the average of a column.
- min()            Calculates the min value of column.
- max()            Calculates the min value of column.
- length()        Calculates the number of characters of a string or the number of characters of the string representation of a number.
- lower()        Returns a string in lower case characters.
- upper()        Returns a string in upper case characters.
- substr()        Returns a sub string.
- typeof()        Returns the data type of a column.

More details can be found in the SQLite documentation here: [Core Functions](#)

and here: [Expressions](#)

and here: [Date And Time Functions](#)

## 2.2.7 Filtering WHERE

After the SELECT expression you can add a WHERE expression for filtering.

The WHERE expression is evaluated for each row in the input data as a Boolean expression. Only rows for which the WHERE clause expression evaluates to true are included from the dataset before continuing. Rows are excluded from the result if the WHERE clause evaluates to either false or NULL.

Some operators used for filtering data:

- = > < >= <=
- AND OR BETWEEN
- LIKE

Examples:

- A single row.  
Where the rowid has the value of the numeric variable idVal  
`SELECT * FROM TableName WHERE rowid = idVal`  
`ResultSet1 = SQL1.ExecQuery("SELECT * FROM TableName WHERE rowid = " & idVal)`  
Where an ID column has the value of the variable idVal  
`SELECT * FROM TableName WHERE ID = idVal`  
`ResultSet1 = SQL1.ExecQuery("SELECT * FROM TableName WHERE ID = " & idVal)`
- A single entry (value).  
`SELECT Col1 FROM TableName WHERE rowid = idVal`  
`Value = SQL1.ExecQuerySingleResult("SELECT Col1 FROM TableName WHERE rowid = idVal")`
- The rows where columns have given values.  
`SELECT * FROM TableName WHERE Col1 LIKE 'abc' AND Col2 LIKE 123`  
`ResultSet1 = SQL1.ExecQuery("SELECT * FROM TableName WHERE Col1 LIKE 'abc%' AND Col2 LIKE 123")`  
The % character can be used as a wildcard:  

abc	means the exact sequence
%abc	means beginning with any characters and ending with abc
abc%	means beginning with abc and ending with any characters
%abc%	means abc anywhere in the string
- The rows where a value in a column is between two given values.  
`SELECT * FROM TableName WHERE Col1 >= minVal AND Col1 <= maxVal`  
`ResultSet1 = SQL1.ExecQuery("SELECT * FROM TableName WHERE Col1 >= minVal AND Col1 <= maxVal")`  
Or with BETWEEN which is the same.  
`SELECT * FROM TableName WHERE Col1 BETWEEN minVal AND maxVal`  
`Cursor1 = SQL1.ExecQuery("SELECT * FROM TableName WHERE Col1 BETWEEN minVal AND maxVal")`  
Or with minVal and maxVal being variables:  
`ResultSet1 = SQL1.ExecQuery("SELECT * FROM TableName WHERE Col1 BETWEEN " & minVal & " AND " & maxVal)`

### 2.2.8 Sorting ORDER BY

If a SELECT statement that returns more than one row does not have an ORDER BY clause, the order in which the rows are returned is undefined.

Or, if a SELECT statement does have an ORDER BY clause, then the list of expressions attached to the ORDER BY determine the order in which rows are returned to the user.

A query can be sorted either ascending or descending.

Add an ORDER BY expression at the end of the query.

- Read the entire database and ordering according to a given column:  
`SELECT * FROM TableName ORDER BY Col1 ASC`                      ascending  
`ResultSet1 = SQL1.ExecQuery("SELECT * FROM TableName ORDER BY Col1 ASC")`

`SELECT * FROM TableName ORDER BY 2 DESC`                      descending  
`ResultSet1 = SQL1.ExecQuery("SELECT * FROM TableName ORDER BY 2 DESC")`

The column to order can be given either by its name Col1 or its number 2.

The column numbering begins with 1.

- Read the given columns and sort on two of them.  
`SELECT FirstName AS [First name], LastName AS [Last name], City FROM persons`  
`ORDER BY LastName ASC, FirstName ASC`  
`ResultSet1 = SQL1.ExecQuery("SELECT FirstName AS [First name], LastName AS [Last name], City FROM persons ORDER BY LastName ASC, FirstName")`  
The brackets [First name] are needed because of the spaces in the alias column names.

### 2.2.9 Date / Time functions

SQLite has no specific Date/Time data type but has several Date/Time functions.

Below the most useful for B4X:

- `date(timestring, modifier, modifier, ...)`  
Returns a date.
- `time(timestring, modifier, modifier, ...)`  
Returns a time.
- `datetime(timestring, modifier, modifier, ...)`  
Returns a date and time.

For more details, examples and what timestring and modifiers are, please look at the [SQLite documentation](#).

In B4X the best way to store dates is to store them as ticks, which are the number of milliseconds since January 1, 1970.

SQLite doesn't have the same ticks but has "unixepoch" ticks which are the number of seconds since January 1, 1970.

Examples of queries with Ticks in the DateTicks column:

DateTicks = 1448795854111 (B4A ticks):

- `ResultSet1 = SQL1.ExecQuery("SELECT date(DateTicks / 1000, 'unixepoch') ...")`  
Returns: 2015-11-29
- `ResultSet1 = SQL1.ExecQuery("SELECT time(DateTicks / 1000, 'unixepoch') ...")`  
Returns: 11:17:34
- `ResultSet1 = SQL1.ExecQuery("SELECT datetime(DateTicks / 1000, 'unixepoch') ...")`  
Returns: 2015-11-29 11:17:34

In these examples `DateTicks / 1000` is the timestring and `'unixepoch'` a modifier.

The date format of the `date()` function is yyyy-MM-dd there is no possibility to change this format with `'unixepoch'`

The `date()` function is used in the SQLiteLight4 example.

Another solution could be to store the date as a String with yyyy-MM-dd or yyyy-MM-dd HH.mm format.

Only these formats must be used if you want to use the functions below.

And when you read the data, you can return it with a format with the *strftime* function.

`strftime(format, timestring, modifier, modifier, ...)`

format can be:

%d	day of month: 00
%f	fractional seconds: SS.SSS
%H	hour: 00-24
%j	day of year: 001-366
%J	Julian day number
%m	month: 01-12
%M	minute: 00-59
%s	seconds since 1970-01-01
%S	seconds: 00-59
%w	day of week 0-6 with Sunday==0
%W	week of year: 00-53
%Y	year: 0000-9999

Examples with Date = '2017-03-09'

- `ResultSet1 = SQL1.ExecQuery("SELECT strftime('%d-%m-%Y', Date) ...")`  
Returns: 09-03-2017
- `ResultSet1 = SQL1.ExecQuery("SELECT strftime('%m-%d-%Y', Date) ...")`  
Returns: 03-09-2017



## 2.2.10 Other functions

### 2.2.10.1 Get the data type of columns typeof()

In a table with a column *Part* of type TEXT *Number* of type INTEGER and another column *Price* of type REAL

```
SELECT typeof(Part), typeof(Number), typeof(Price) FROM TableName
```

```
ResultSet1 = SQL1.ExecQuery("SELECT typeof(Part), typeof(Number), typeof(Price) FROM  
TableName")
```

Get the data type with:

Column	request	or	other request	>	result
Part:	Cursor1.GetString("Part")	or	Cursor1.GetString2(0)	>	text
Number:	Cursor1.GetString("Number")	or	Cursor1.GetString2(1)	>	integer
Price:	Cursor1.GetString("Price")	or	Cursor1.GetString2(2)	>	real

### 2.2.10.2 Get the max length of the data in a column length()

For a string, the returned value is the number of characters not the number of bytes.

For a number, the returned value is the number of characters of its string representation.

For a blob, the returned value is the number of bytes.

```
SELECT max(length(Col1)) FROM TableName
```

```
MaxChars = SQL1.ExecQuerySingleResult("SELECT max(length(Col1)) FROM  
TableName")
```

### 2.2.10.3 Get a sub string substr()

The substr(String, BeginIndex, Lenght) function returns a sub sting of *String* beginning with the character at the *BeginIndex* position and with the number of characters given in *Lenght*.

If Lenght is omitted, substr returns the sub string from BeginIndex to the end of the string.

The index of the first character is 1.

Example:

Get the year from a date string '31/11/2016'

```
SELECT substr(Date, 7, 4) AS Year FROM TableName
```

```
ResultSet1 = SQL1.ExecQuery("SELECT substr(Date,7,4) AS Year FROM TableName")
```

Retrieve a date into another format:

Change a date from YYYY-MM-DD format to DD.MM.YYYY format:

```
ResultSet1 = SQL1.ExecQuery("SELECT substr(Date,9,2) || '.' || substr(Date,6,2) || '.'  
|| substr(Date,1,4) AS Date FROM TableName")
```

Is || the concatenate operator.

|| '.' || concatenates two strings with a dot . inbetween.

#### 2.2.10.4 Replace parts of a string replace()

The `replace(String, Target, Replace)` function returns a string formed by substituting string *Replace* for every occurrence of string *Target* in *String*. The `replace` function is case sensitive. Equivalent to `MyText.Replace(SubString)` in B4X.

Example:

In a date like 2016-12-31 replace '-' by '/' to get 2016/12/31

```
ResultSet1 = SQL1.ExecQuery("SELECT replace(Date,'-', '/') AS Date FROM TableName")
```

#### 2.2.10.5 Find a substring in a string instr()

The `instr(String, SubString)` function finds the first occurrence of *SubString* within *String* and returns the number of prior characters plus 1, or 0 if *SubString* is nowhere found within *String*. Equivalent to `MyText.IndexOf(SubString)` in B4X.

Example:

```
ResultSet1 = SQL1.ExecQuery("SELECT instr(Date,'2016') AS New FROM TableName")
```

#### 2.2.10.6 Round a number round()

The `round(Number, Digits)` function returns a floating-point value *Number* rounded to *Digits* digits to the right of the decimal point. If the *Digits* argument is omitted, it is assumed to be 0. Equivalent to `Round2(Number, Digits)` in B4X.

Example:

Gets the value in the column *Number* and rounds it to two decimals and sets the column alias to *Value*.

```
ResultSet1 = SQL1.ExecQuery("SELECT round(Number,2) AS Value FROM TableName")
```

#### 2.2.10.7 Get the total number of rows count()

`SELECT count() FROM TableName`

```
NumberOfRows = SQL1.ExecQuerySingleResult("SELECT count() FROM TableName")
```

#### 2.2.10.8 Get the tables in the database sqlite\_master

`SELECT name FROM sqlite_master WHERE Type='table'`

```
ResultSet1 = SQL1.ExecQuery("SELECT name FROM sqlite_master Where Type='table'")
```

#### 2.2.10.9 Get the column names of a table TableName

`SELECT * FROM TableName`

```
ResultSet1 = SQL1.ExecQuery("SELECT * FROM TableName")
```

`Do While` `ResultSet1.NextRow`

```
    ColumnName(i) = ResultSet1.GetColumnName(i)
```

`Loop`

**2.2.10.10 Get the number of database rows that were changed changes()**

Get the number of database rows that were changed or inserted or deleted by the most recently completed INSERT, DELETE, or UPDATE.

```
SELECT changes() FROM TableName
```

```
NbChanges = SQL1.ExecQuerySingleResult("SELECT changes() FROM TableName")
```

**2.2.10.11 Get the PRIMARY KEYs from a table rowid**

Get the PRIMARY KEYs from a table and save them in a List, *rowid* is a reserved column name.

This is valid even if there is no column defined with PRIMARY KEY.

This function throws an error if the table is empty!

```
SELECT rowid FROM TableName
```

```
Private IDList As List
```

```
Private ResultSet1 As ResultSet
```

```
IDList.Initialize
```

```
ResultSet1 = SQL1.ExecQuery("SELECT rowid FROM TableName")
```

```
Do While ResultSet1.NextRow
```

```
    IDList.Add(Cursor1.GetLong2(0))
```

```
Loop
```

**2.2.10.12 Get the last insert rowid from a table last\_insert\_rowid**

Get the *rowid* of the last insert data set.

```
SELECT last_insert_rowid FROM TableName
```

```
Private LastRowID As Long
```

```
LastRowID = SQL1.ExecQuerySingleResult("SELECT last_insert_rowid() FROM TableName")
```

### 2.2.11 ResultSet GetInt, GetInt2 etc B4A, B4i, B4J

To read data from a database we use the ResultSet object.

We use a while / Do loop and ResultSet1.NextRow to go through the rows.

Examples:

Reads the rowids into a List:

```
Private ResultSet1 As ResultSet
ResultSet1 = SQL1.ExecQuery("SELECT rowid FROM persons")
RowIDList.Initialize

Do While ResultSet1.NextRow
    RowIDList.Add(ResultSet1.GetInt2(0))
Loop
```

Fills three ListViews with data:

```
Private ResultSet1 As ResultSet
ResultSet1 = SQL1.ExecQuery("SELECT FirstName, LastName, City FROM persons")
lrvFirstName.Clear
lrvLastName.Clear
lrvCity.Clear

Do While ResultSet1.NextRow
    lrvFirstName.Add(ResultSet1.GetString2(0))
    lrvLastName.Add(ResultSet1.GetString2(1))
    lrvCity.Add(ResultSet1.GetString2(2))
Loop
```

The following methods extract the different data from the Cursor.

- **ResultSet.GetInt** returns an Integer value.
- **ResultSet.GetLong** returns a Long value.
- **ResultSet.GetDouble** returns a Double value.
- **ResultSet.GetString** returns a String value.
- **ResultSet.GetBlob** returns a **Binary Large Object**, used for images.

For each method two version exist:

**ResultSet.GetXXX(ColumnName)** / **ResultSet.GetXXX2(ColumnIndex)**

ColumnName must be either:

- The name as defined in the table creation (case sensitive).
- The alias name defined in the query.

ColumnIndex is the column index in the query (beginning with 0).

From the example:

ResultSet.GetString(FirstName)

ResultSet.GetString2(0)

### 2.2.12 Cursor GetInt, GetInt2 etc B4A only

**The Cursor object exists only in B4A.**

**I suggest to not use it !**

The Cursor object holds the result data form a query.

The data is organized in rows each row contains the data for each column defined in the query.

Example:

```
Cursor1 = Starter.SQLite1.ExecQuery("SELECT FirstName As [First name], LastName As [Last name], City FROM persons")
```

Each row holds 3 values, one for each column.

First, we need to set the index of the row with:

**Cursor.Position** =RowIndex sets the row index, the row count begins with 0.

```
Cursor1.Position = 0
```

We get the number of rows with:

**Cursor.RowCount** returns the number of rows, Cursor.RowCount = 0 if no result is found.

```
RowNb = Cursor1.RowCount
```

We get the number of columns with:

**Cursor.ColumnCount** returns the number of columns.

```
ColNb = Cursor1.ColumnCount
```

The following methods extract the different data from the Cursor.

- **Cursor.GetInt** returns an Integer value.
- **Cursor.GeLong** returns a Long value.
- **Cursor.GetDouble** returns a Double value.
- **Cursor.GetString** returns a String value.
- **Cursor.GetBlob** returns a **Binary Large Object**, used for images.

For each method two version exist:

**Cursor.GetXXX(ColumnName)** / **GetXXX2(ColumnIndex)**

ColumnName must be either:

- The name as defined in the table creation.
- The alias name defined in the query.

ColumnIndex is the column index in the query.

From the example:

```
Cursor.GetString(FirstName)
```

```
Cursor.GetString2(0)
```

### 2.2.13 Get Table information PRAGMA

It uses a special query PRAGMA.

This query returns one row per column with following data :

Column index name    Explanation

- 0    cid                      column index
- 1    name                    column name
- 2    type                    data type
- 3    dflt\_value            default value
- 4    notnull                null if the database accepts null values
- 5    pk                      primary key = 1 if the column is a PRIMARY KEY otherwise = 0  
This is valid only if a column with a primary key was created.

Example using the column indexes:

```
ResultSet1 = SQL1.ExecQuery("PRAGMA table_info (TableName)")
Do While ResultSet1.NextRow
    For j = 0 To ResultSet1.ColumnCount - 1
        Log(i & " / " & j & " : " & ResultSet1.GetString2(j))
    Next
    Log(" ")
Loop
```

Or this code, using the column names:

```
ResultSet1= SQL1.ExecQuery("PRAGMA table_info (TableName)")
Do While ResultSet1.NextRow
    Log("ID : " & ResultSet1.GetString("cid"))
    Log("Name : " & ResultSet1.GetString("name"))
    Log("Type : " & ResultSet1.GetString("type"))
    Log("Default value : " & ResultSet1.GetString("dflt_value"))
    Log("Not null : " & ResultSet1.GetString("notnull"))
    Log("Primary key : " & ResultSet1.GetString("pk"))
    Log(" ")
Loop
```

**2.2.14 Deleting data                      DELETE FROM**

DELETE FROM TableName WHERE ID = idVal

```
SQL1.ExecNonQuery("DELETE FROM TableName WHERE ID = idVal")
```

**2.2.15 Rename a table                      ALTER TABLE Name RENAME TO**

Renames a given table.

ALTER TABLE TableName RENAME TO NewTableName)

```
SQL1.ExecNonQuery("ALTER TABLE TableName RENAME TO NewTableName")
```

**2.2.16 Add a column                      ALTER TABLE Name ADD COLUMN**

Add a new column to the database.

ALTER TABLE TableName ADD COLUMN Colname ColType)

```
SQL1.ExecNonQuery("ALTER TABLE TableName ADD COLUMN ColN TEXT")
```

**2.2.16.1            Update the database after having added a column**

Update the database after having added a new column.

- Sets the values of all rows in the new column to an empty string.

UPDATE TableName SET ColName = "

```
SQL1.ExecNonQuery("UPDATE TableName SET ColN = ''")
```

- Sets the values of the rows in a column to a given new value where the value is another old value.

UPDATE TableName SET ColName = 'ValueNew' WHERE ColName = 'ValueOld'

```
SQL1.ExecNonQuery("UPDATE TableName SET ColN = 'ValueNew' WHERE ColN = 'ValueOld'")
```

**2.2.17 Delete a table                      DROP TABLE**

The DROP TABLE statement removes a table added with the CREATE TABLE statement. The name specified is the table name. The dropped table is completely removed from the database schema and the disk file. The table cannot be recovered. All indices and triggers associated with the table are also deleted.

The optional IF EXISTS clause suppresses the error that would normally result if the table does not exist.

DROP TABLE IF EXISTS TableName

```
SQL1.ExecNonQuery("DROP TABLE IF EXISTS TableName")
```

### 2.2.18 Insert an image

To insert an image, we need a BLOB (Binary Large Object).  
The column type in the database must be set to BLOB !

```
Sub InsertBlob
    'convert the image file to a bytes array
    Private InputStream1 As InputStream
    InputStream1 = File.OpenInput(File.DirAssets, "smiley.gif")
    Private OutputStream1 As OutputStream
    OutputStream1.InitializeToByteArray(1000)
    File.Copy2(InputStream1, OutputStream1)
    Private Buffer() As Byte 'declares an empty array
    Buffer = OutputStream1.ToByteArray

    'write the image to the database
    SQL1.ExecNonQuery2("INSERT INTO table2 VALUES('smiley', ?)", Array As String(Buffer))
End Sub
```

Here we are using a special type of OutputStream which writes to a dynamic byte array.  
File.Copy2 copies all available data from the input stream into the output stream.  
Then the bytes array is written to the database.

### 2.2.19 Read an image

Using a ResultSet.GetBlob we fetch the stored image.  
Now we are using an input stream that reads from this array and load the image.

```
Sub ReadBlob
    Private ResultSet1 As ResultSetr
    'Using ExecQuery2 is safer as it escapes special characters automatically.
    'In this case it doesn't really matter.
    ResultSet1 = SQL1.ExecQuery2("SELECT image FROM table2 WHERE name = ?", Array As
String("smiley"))
    ResultSet1.NextRow
    Private Buffer() As Byte 'declare an empty byte array
    Buffer = ResultSet1.GetBlob("image")
    Private InputStream1 As InputStream
    InputStream1.InitializeFromByteArray(Buffer, 0, Buffer.Length)

    Private Bitmap1 As Bitmap
    Bitmap1.Initialize2(InputStream1)
    InputStream1.Close
End Sub
```



### 2.2.20 ExecQuery vs ExecQuery2 / ExecNonQuery vs ExecNonQuery2

The examples below suppose a table with three columns:  
Col1 TEXT, Col2 INTEGER, Col3 INTEGER

There are two ways to execute a query.

- **ExecQuery**(Query As String)

Executes the query, you must take care of the datatype.

Example:

```
ResultSet1 = SQL1.ExecQuery("SELECT * FROM TableName WHERE Col1 = '" & MyText &  
"'" AND Col2 >= " & minVal & " AND Col2 <= " & maxVal)
```

Note that MyText is between two quotes because the data field is a TEXT field!

- **ExecQuery2**(Query As String, StringArgs As String())

The query includes question marks which will be replaced with the values in the array.

Example:

```
ResultSet1 = SQL1.ExecQuery2("SELECT * FROM TableName WHERE Col1 = ? AND Col2 >=  
? AND Col2 <= ? ", Array As String (MyText, minVal, maxVal))
```

Note that ExecQuery2 is safer because it takes care of the column data type!

Note that with ExecQuery and text, you need to put the text between quotes like 'text'.  
With ExecQuery2 and text, you must not use the quotes, ExecQuery2 takes care of it.

The same for ExecNonQuery.

- **ExecNonQuery**(Query As String)

Executes the query, you must take care of the datatype.

Example:

```
SQL1.ExecNonQuery("INSERT INTO table1 VALUES('abc', 1, 2)")
```

Note that abc is between two quotes because the data field is a TEXT field!

- **ExecNonQuery2**(Query As String, StringArgs As String())

The query includes question marks which will be replaced with the values in the array.

Example:

```
SQL1.ExecNonQuery2("INSERT INTO table1 VALUES(?, ?, ?)", Array As String("abc",  
3, 4))
```

Note that ExecQuery2 is safer because it takes care of the column data type!

The same exists for ExecQuerySingleResult and ExecQuerySingleResult2.

### 2.2.21 Insert many rows SQL.BeginTransaction / SQL.EndTransaction

```

Sub InsertManyRows
    SQL1.BeginTransaction
    Try
        For i = 1 To 500
            SQL1.ExecNonQuery2("INSERT INTO table1 VALUES ('def', ?, ?)", Array As String(i,
i))
        Next
        SQL1.TransactionSuccessful
    Catch
        Log(LastException.Message)
    End Try
    SQL1.EndTransaction
End Sub

```

This code is an example of adding many rows. Internally a lock is acquired each time a "writing" operation is done.

By explicitly creating a transaction the lock is acquired once.

The above code took less than half a second to run on a real device.

Without the BeginTransaction / EndTransaction block it took about 70 seconds.

A transaction block can also be used to guarantee that a set of changes were successfully done.

Either all changes are made, or none are made.

By calling SQL.TransactionSuccessful we are marking this transaction as a successful transaction.

If you omit this line, all the 500 INSERTS will be ignored.

It is very important to call EndTransaction eventually.

Therefore, the transaction block should usually look like:

```

SQL1.BeginTransaction
Try
    'Execute the sql statements.
SQL1.TransactionSuccessful
Catch
    'the transaction will be cancelled
End Try
SQL1.EndTransaction

```

### 2.2.22 Asynchronous queries

The SQL library supports asynchronous select queries and asynchronous batch inserts.

Asynchronous means that the task will be processed in the background and an event will be raised when the task completes. This is useful when you need to issue a slow query and keep your application responsive.

The usage is quite simple:

```
SQL1.ExecQueryAsync("SQL1", "SELECT * FROM table1", Null)
...
Sub SQL1_QueryComplete (Success As Boolean, ResultSet1 As ResultSet)
    If Success Then
        Do While ResultSet1.NextRow
            Log(ResultSet1.GetInt2(0))
        Loop
    Else
        Log(LastException)
    End If
End Sub
```

The first parameter is the "event name". It determines which sub will handle the QueryComplete event.

#### Since B4A 7.00, B4i 4.00 and B4J 5.50

you could use this code using a resumable sub and Wait For.

```
Dim rs as ResultSet
Dim SenderFilter As Object = SQL1.ExecQueryAsync("SQL", "SELECT * FROM table1", Null)
Wait For (SenderFilter) SQL_QueryComplete (Success As Boolean, rs As ResultSet)
If Success Then
    Do While rs.NextRow
        Log(rs.GetInt2(0))
    Loop
    rs.Close
Else
    Log(LastException)
End If
```

### 2.2.23 Batch inserts AddNonQueryToBatch / ExecNonQueryBatch

SQL.AddNonQueryToBatch / ExecNonQueryBatch allow you to asynchronously process a batch of non-query statements (such as INSERT statements).

You should add the statements by calling AddNonQueryToBatch and eventually call ExecNonQueryBatch.

The task will be processed in the background. The NonQueryComplete event will be raised after all the statements execute.

```
For i = 1 To 10000
    SQL1.AddNonQueryToBatch("INSERT INTO table1 VALUES (?)", Array As String(Rnd(0,
100000)))
Next
SQL1.ExecNonQueryBatch("SQL")

...
Sub SQL_NonQueryComplete (Success As Boolean)
    Log("NonQuery: " & Success)
    If Success = False Then Log(LastException)
End Sub
```

Since B4A 7.00, B4i 4.00 and B4J 5.50 you should use this code using a resumable sub.

```
For i = 1 To 1000
    SQL1.AddNonQueryToBatch("INSERT INTO table1 VALUES (?)", Array(Rnd(0, 100000)))
Next
Dim SenderFilter As Object = SQL1.ExecNonQueryBatch("SQL")
Wait For (SenderFilter) SQL_NonQueryComplete (Success As Boolean)
Log("NonQuery: " & Success)
```

## 2.3 Multiple tables

A database can, of course, have more than one table.

Example: This is only a small simple example to demonstrate the principle.  
Demo code example project SQLiteLight4.

Database with 3 tables:

- |             |                                       |                                    |                                   |
|-------------|---------------------------------------|------------------------------------|-----------------------------------|
| • Stock     | Number INTEGER,<br>number of products | ProductID INTEGER,<br>product ID   | Date INTEGER<br>date in Ticks     |
| • Products  | Name TEXT,<br>product name            | Price REAL,<br>product price       | SupplierID INTEGER<br>supplier ID |
| • Suppliers | Name TEXT,<br>suppliers name          | Address TEXT,<br>suppliers address | City TEXT<br>suppliers city       |

In the table Stock we use the ID of the product rather than its name.  
The same for the Supplier in the Products table.

Query example of a call for display:

```
Query = "SELECT Stock.Number, Products.Name AS Product, Suppliers.Name AS Supplier,
Products.Price AS Price, Stock.Number * Products.Price AS Value, date(Stock.Date /
1000, 'unixepoch') AS Date"
Query = Query & " FROM Stock, Products, Suppliers"
Query = Query & " WHERE Products.rowid = Stock.ProductID AND Suppliers.rowid =
Products.SupplierID"
```

We want to read following data:

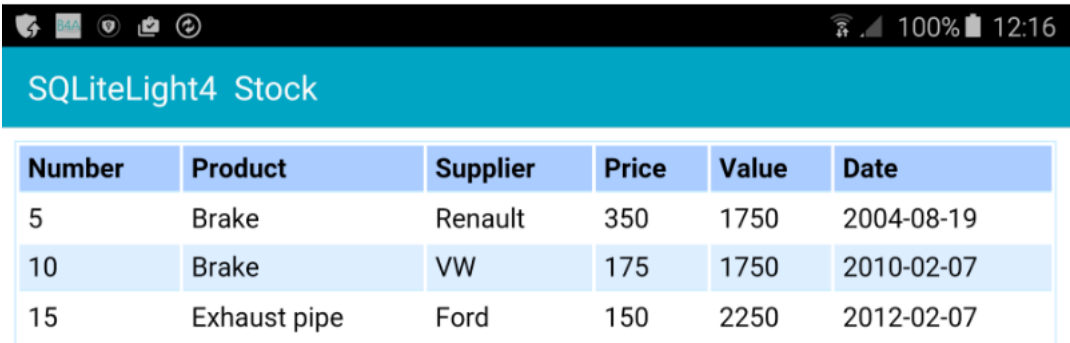
- The number of items in stock `Stock.Number`.  
The Number column in the Stock table.
- The product name `Products.Name AS Product`.  
The Name column in the Products table and give this column the name 'Product'.
- The supplier name `Suppliers.Name AS Supplier`.  
The Name column in the Suppliers tabel and give this column the name 'Supplier'.
- The product price `Products.Price AS Price`.  
The Price column in the Products table and give this column the name 'SPrice'.
- The value of these products in stock `Stock.Number * Products.Price AS Value`.  
The multiplication of the number of items in stock with the product price and give this column the name 'Value'.
- The date when the product was entered `date(Stock.Date / 1000, 'unixepoch') AS Date`.  
We use the SQLite date function where we give the Date column of the Stock table.  
As the date is in B4A ticks we need to devide the value by 1000 to adapt it to 'SQL ticks'  
and we must add the parameter 'unixepoch' for 'SQL ticks'.

The query involves the three tables Stock, Products and Suppliers:  
`FROM Stock, Products, Suppliers`

We must add a WHERE expression:

- To connect the Products table rowid to the Stock ProductID column value.  
`Products.rowid = Stock.ProductID`
- To connect the Suppliers table rowid to the Products SupplierID column value.  
`Suppliers.rowid = Products.SupplierID`

Example of the result:



Number	Product	Supplier	Price	Value	Date
5	Brake	Renault	350	1750	2004-08-19
10	Brake	VW	175	1750	2010-02-07
15	Exhaust pipe	Ford	150	2250	2012-02-07

For more details, look at the SQLiteLight4 example program.

## 2.4 Transaction speed

If you have to do many inserts into a database, you should use BeginTransaction and EndTransaction this will considerably speed up the process.

A transaction is a set of multiple "writing" statements that are automatically committed.

It is particularly important to handle transactions carefully and close them.

The transaction is considered successful only if TransactionSuccessful is called. Otherwise no changes will be made.

Typical usage:

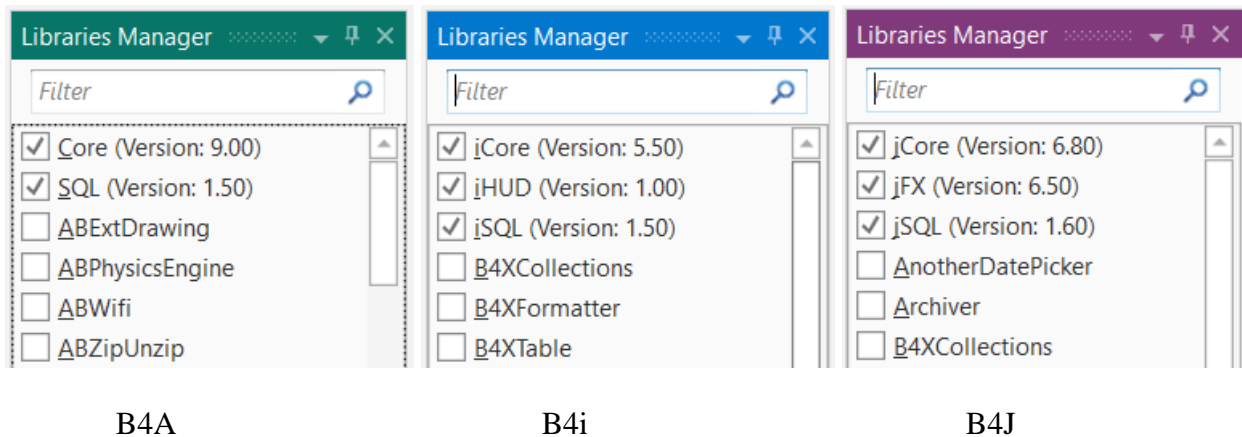
```
SQL1.BeginTransaction
Try
    'block of statements like:
    For i = 1 To 1000
        SQL1.ExecNonQuery("INSERT INTO table1 VALUES(...)")
    Next
    SQL1.TransactionSuccessful
Catch
    Log(LastException.Message) 'no changes will be made
End Try
SQL1.EndTransaction
```

## 2.5 First steps

To use a database, we must:

### 2.5.1 Reference the SQLite library

First reference the SQL library in the Libs Tab in the lower right corner in the IDE.



### 2.5.2 Declare the SQLite library

Declare it with `Public` in the `Process_Globals` routine of the Main module or for B4A in the `Process_Globals` routine of the Starter Service module.

I suggest, to define two other variables for the database path and file name:

```
Sub Process_Globals
    Public SQL1 As SQL
    Public SQLDataBasePath As String
    Public SQLDataBaseName As String
```

B4A: In the Starter Module or in the Main module if you use only one Activity.

B4i and B4J: In the Main module.



### 2.5.3 Initialize the SQLite library and the variables

Set values to the variables and initialize the SQLite library.

The value of `SQLDataBasePath` will be different on the three operating systems.

Then it depends on if you generate a database in the code or if you copy a database from `File.DirAssets` if it doesn't exist.

Example for B4A.

```
SQLDataBasePath = File.DirInternal
SQLDataBaseName = "persons.db"

' File.Delete(SQLDataBasePath, SQLDataBaseName) ' for testing, removes the database

'check if the database already exists
If File.Exists(File.DirInternal, SQLDataBaseName) = False Then
    'if not, initialize it
    'copy the default DB
    File.Copy(File.DirAssets, SQLDataBaseName, SQLDataBasePath, SQLDataBaseName)
    'initialize it
    SQL1.Initialize(SQLDataBasePath, SQLDataBaseName, True)
    'or create it
    'CreateDataBase
Else
    'if yes, initialize it
    SQL1.Initialize(File.DirInternal, "persons.db", True)
End If
```

#### B4A

Initialize it in the `Service_Create` routine in the Starter Service.

Or in the Main module with `If FirstTime Then / End If`

If you already have a database in the Files folder of the project you need to copy it from `File.DirAssets` in another folder.

Databases are NOT accessible from `File.DirAssets` even for reading only!

```
Sub Starter_Create(FirstTime As Boolean)
    SQLDataBasePath = File.DirInternal
    SQLDataBaseName = "persons.db"

    ' File.Delete(File.DirInternal, "persons.db") ' only for testing, removes the
    database

    'check if the database already exists
    If File.Exists(SQLDataBasePath, SQLDataBaseName) = False Then
        'if not, initialize it
        SQL1.Initialize(SQLDataBasePath, SQLDataBaseName, True)
        'and create it
        CreateDataBase
    Else
        'if yes, initialize it
        SQL1.Initialize(SQLDataBasePath, SQLDataBaseName, True)
    End If
```

**B4i**

In the **Application\_Start** routine:

```
Private Sub Application_Start (Nav As NavigationController)
    NavControl = Nav
    Page1.Initialize("Page1")
    Page1.RootPanel.LoadLayout("main")
    NavControl.ShowPage(Page1)

    SQLDataBasePath = File.DirDocuments
    SQLDataBaseName = "persons.db"

    ' File.Delete(SQLDataBasePath, "persons.db") ' only for testing, removes the database

    'check if the database already exists
    If File.Exists(SQLDataBasePath, SQLDataBaseName) = False Then
        'if not, initialize it
        SQL1.Initialize(SQLDataBasePath, SQLDataBaseName, True)
        'and create it
        CreateDataBase
    Else
        SQL1.Initialize(SQLDataBasePath, SQLDataBaseName, True)
    End If
```

**B4J**

In the #Region Project Attributes you must reference the SQLite jar file :

```
#Region Project Attributes
#MainFormWidth: 600
#MainFormHeight: 800
#AdditionalJar: sqlite-jdbc-3.7.2
#End Region
```

In the **AppStart** routine:

```
Sub AppStart (Form1 As Form, Args() As String)
    MainForm = Form1
    MainForm.RootPane.LoadLayout("Main") 'Load the layout file.
    MainForm.Show

    SQLDataBasePath = File.DirData("jSQLiteLight1")
    SQLDataBaseName = "persons.db"

    ' File.Delete(File.DirData("jSQLiteLight1"), "persons.db") ' only for testing,
    removes the database

    'check if the database already exists
    If File.Exists(SQLDataBasePath, SQLDataBaseName) = False Then
        'if not, initialize it
        SQL1.InitializeSQLite(SQLDataBasePath, SQLDataBaseName, True)
        'and create it
        CreateDataBase
    Else
        'if yes, initialize it
        SQL1.InitializeSQLite(SQLDataBasePath, SQLDataBaseName, True)
    End If
```

## 2.6 SQLite Database first simple example program SQLiteLight1

There are three projects, one for each operating system.

This example programs are very simple projects with a very simple user interface. The goal is to show how to use SQLite with as much as possible the same code for the three operating systems and not optimizing the layouts nor use operating system specific layouts.

The source codes are located in these folders:

*SQLiteDatabase\_SourceCode\SQLiteLight1\B4A*

*SQLiteDatabase\_SourceCode\SQLiteLight1\B4i*

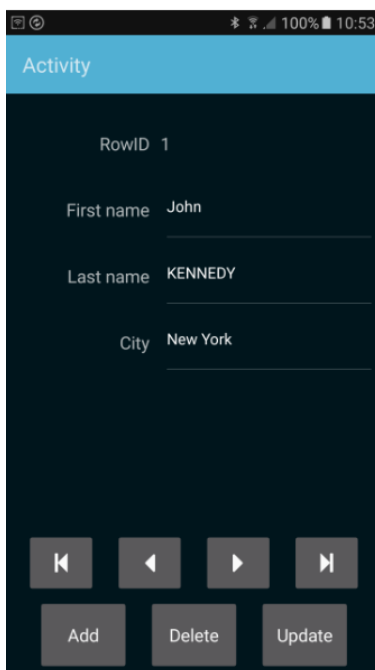
*SQLiteDatabase\_SourceCode\SQLiteLight1\B4J.*

The database name, the table name and the column names are hard coded, to make the code better readable.

At the first run the program generates a new example database.

Following functions are implemented:

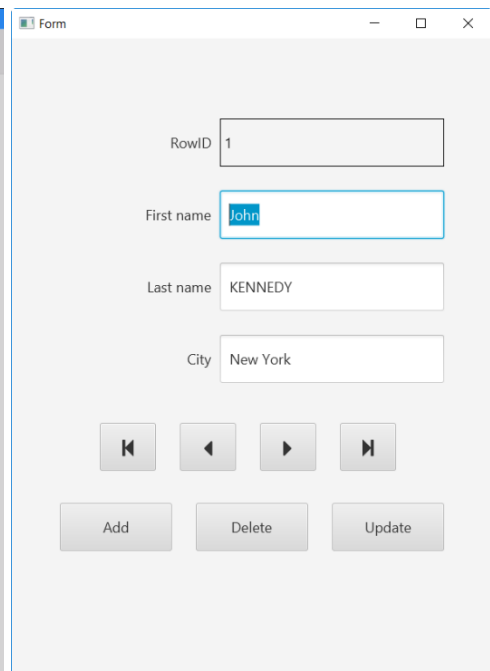
- Add / Edit an entry
- Delete an entry
- Update an entry
- Display first, previous, next and last entry.



B4A



B4i



B4J

## 2.6.1 Source code

It is self-explanatory.

### 2.6.1.1 B4A Program initialization

#### 2.6.1.1.1 Process\_Global

We dim the process global variables.

```
Sub Process_Globals
    Public SQL1 As SQL

    Public CurrentIndex = -1 As Int    ' index of the current entry

    Public RowIDList As List    ' list containing the IDs of the database
    ' we need it because the rowids can be different from the list indexes
    ' if we delete an entry its rowid is lost
End Sub
```

#### 2.6.1.1.2 Globals

We dim all the views of the layout.

```
Sub Globals
    Private lblRowID As Label
    Private edtFirstName, edtLastName, edtCity As EditText
End Sub
```

#### 2.6.1.1.3 Activity\_Create

We check if the database already exists, initialize it, load it, and show the first entry.

```
Sub Activity_Create(FirstTime As Boolean)
    '**** opreating system specific code
    Activity.LoadLayout("Main")
    Activity.Title = "SQLiteLight1"

    '**** program specific code
    If FirstTime Then
        ' File.Delete(File.DirInternal, "persons.db") ' for testing, removes the database

        'check if the database already exists
        If File.Exists(File.DirInternal, "persons.db") = False Then
            'if not, initialize it
            SQL1.Initialize(File.DirInternal, "persons.db", True)
            'and create it
            CreateDataBase
        Else
            'if yes, initialize it
            SQL1.Initialize(File.DirInternal, "persons.db", True)
        End If
    End If
End Sub
```

#### 2.6.1.1.4 Activity\_Resume

If the database is not initialized, we initialize it, initialize the IDList list, read the database and show the first entry.

```
Sub Activity_Resume
    RowIDList.Initialize      'initialize the ID list
    ReadDataBase              'read the database
    ShowEntry(0)              'show the first entry
End Sub
```

#### 2.6.1.1.5 Activity\_Pause

##### Program closing:

If the program is closed by the user, we close the database.

```
Sub Activity_Pause (UserClosed As Boolean)
    If UserClosed Then
        SQL1.Close           'if the user closes the program we close the database
    End If
End Sub
```

### 2.6.1.2 B4i Program initialisation

#### 2.6.1.2.1 Process\_Globals

We dim the process global variables.

```
Sub Process_Globals
    Public App As Application
    Public NavControl As NavigationController
    Private Page1 As Page
    Private HUD1 As HUD          ' HUD library, used to display Toastmessages

    Public lblRowID As Label
    Public edtFirstName, edtLastName, edtCity As TextField

    Public SQL1 As SQL

    Private CurrentIndex As Int  ' index of the current entry

    Public RowIDList As List     ' list containing the RowIDs of the database
    ' we need it because the IDs can be different from the list indexes
    ' if we delete an entry its ID is lost
End Sub
```

#### 2.6.1.2.2 Application\_Start

```
Private Sub Application_Start (Nav As NavigationController)
    '**** opreating system specific code
    NavControl = Nav
    Page1.Initialize("Page1")
    Page1.Title = "iSQLiteLight1"
    Page1.RootPanel.Color = Colors.White
    Page1.RootPanel.LoadLayout("main")
    NavControl.ShowPage(Page1)

    '**** program specific code
    ' File.Delete(File.DirDocuments, "persons.db") ' for testing, removes the database

    'check if the database already exists
    If File.Exists(File.DirDocuments, "persons.db") = False Then
        'if not, initialize it
        SQL1.Initialize(File.DirDocuments, "persons.db", True)
        'and create it
        CreateDataBase
    Else
        SQL1.Initialize(File.DirDocuments, "persons.db", True)
    End If

    RowIDList.Initialize 'initialize the RowID list
End Sub
```

#### 2.6.1.2.3 Page\_Resize

```
Private Sub Page1_Resize(Width As Int, Height As Int)
    ' read the database
    ReadDataBase
    CurrentIndex = 0
    ShowEntry(CurrentIndex)
End Sub
```

### 2.6.1.3 B4J Program initialisation

#### 2.6.1.3.1 Process\_Globals

We dim the process global variables.

```
Sub Process_Globals
    'operating system object
    Private fx As JFX
    Private MainForm As Form
    Private lblToastMessage As Label
    Public ToastMessageTimer As Timer

    'program specific objects and variables
    Public lblRowID As Label
    Public edtFirstName, edtLastName, edtCity As TextField
    Public SQL1 As SQL

    Private CurrentIndex = 0 As Int      ' index of the current entry

    Public RowIDList As List              ' list containing the RowIDs of the database
    ' we need it because the IDs can be different from the list indexes
    ' if we delete an entry its ID is lost
End Sub
```

#### 2.6.1.3.2 AppStart

```
Sub AppStart (Form1 As Form, Args() As String)
    '**** operating system specific code
    MainForm = Form1
    MainForm.RootPane.LoadLayout("Main") 'Load the layout file.
    MainForm.Show
    MainForm.Title = "jSQLiteLight1"

    ToastMessageTimer.Initialize("ToastMessageTimer", 1000)

    '**** program specific code
    ' File.Delete(File.DirData("jSQLiteLight1"), "persons.db") ' for testing, removes the
    database

    'check if the database already exists
    If File.Exists(File.DirData("jSQLiteLight1"), "persons.db") = False Then
        'if not, initialize it
        SQL1.InitializeSQLite(File.DirData("jSQLiteLight1"), "persons.db", True)
        'and create it
        CreateDataBase
    Else
        'if yes, initialize it
        SQL1.InitializeSQLite(File.DirData("jSQLiteLight1"), "persons.db", True)
    End If

    RowIDList.Initialize
    'initialize the ID list

    ' read the databases
    ReadDataBase
    CurrentIndex = 0
    ShowEntry(CurrentIndex)
End Sub
```



## 2.6.2 Database handling

### 2.6.2.1 Create database

We create the 'persons' table with the three following columns:

- FirstName the persons first name with TEXT data type.
- LastName the persons last name with TEXT data type.
- City the city where the person is living with TEXT data type.

**Same code for all three operating systems!**

```
Private Sub CreateDataBase
    Private Query As String

    'create the database with 3 columns
    Query = "CREATE TABLE persons (FirstName TEXT, LastName TEXT, City TEXT)"
    SQL1.ExecNonQuery(Query)

    'Fill a few entries
    Query = "INSERT INTO persons VALUES (?, ?, ?)"
    SQL1.ExecNonQuery2(Query, Array As String ("John", "KENNEDY", "New York"))
    SQL1.ExecNonQuery2(Query, Array As String ("Peter", "FALK", "Los Angeles"))
    SQL1.ExecNonQuery2(Query, Array As String ("Jack", "LONDON", "Seattle"))
    SQL1.ExecNonQuery2(Query, Array As String ("Ronald", "REGAN", "Los Angeles"))
End Sub
```

### 2.6.2.2 ReadDataBase

We

- Define a ResultSet and read the rowids from the database.
- Check if the database is not empty.
- Fill IDList with the rowids of all entries.
- Set the current index to 0
- Close the ResultSet

Why do we use a List with the rowids?

We use for the ID the rowid which is unique.

If we delete an entry its rowid is lost, which means that the rowid numbers are not simply the row indexes but there can be 'holes' in the list.

**Same code for all three operating systems!**

```
Private Sub ReadDataBase
    Private ResultSet1 As ResultSet

    RowIDList.Clear 'initialize the RowID list
    'We read only the internal 'rowid' column and put rowids in a List
    ResultSet1 = SQL1.ExecQuery("SELECT rowid FROM persons")
    Do While ResultSet1.NextRow
        RowIDList.Add(ResultSet1.GetInt2(0)) 'add the rowid's to the RowID list
    Loop
    CurrentIndex = 0 'set the current index to 0
    ResultSet1.Close 'close the ResultSet, we don't need it anymore
End Sub
```

### 2.6.2.3 ShowEntry

We get the selected entries rowid from IDList, read the entry from the database, fill the EditText views and close the ResultSet.

**Same code for all three operating systems!**

```
Private Sub ShowEntry(EntryIndex As Int)
    Private ResultSet1 As ResultSet
    Private RowID As Int

    If RowIDList.Size = 0 Then      'check if the database is empty
        Return                    'if yes leave the routine
    End If

    RowID = RowIDList.Get(EntryIndex) 'get the RowID for the given entry index
    'read the entry with the given RowID
    ResultSet1 = SQL1.ExecQuery("SELECT * FROM persons WHERE rowid = " & RowID)
    lblRowID.Text = RowID           'display the RowID
    ResultSet1.NextRow              'set the next row
    edtFirstName.Text = ResultSet1.GetString("FirstName") 'read the FirstName column
    edtLastName.Text = ResultSet1.GetString("LastName")  'read the LastName column
    edtCity.Text = ResultSet1.GetString("City")           'read the City column
    ResultSet1.Close                'close the ResultSet, we don't it anymore
End Sub
```

### 2.6.2.4 AddEntry

We first check if an entry with the same name already exists.

If yes, we display a message.

If not, we add the new entry.

Display the new entries ID.

Close the ResultSet.

We use ExecQuery2 instead of ExecQuery, it's easier because we don't need to take care of the data type, the routine converts the data to the correct SQLite type.

The ? sign is a placeholder for the data which must be given in the array.

```
Private Sub AddEntry
    Private Query As String
    Private ResultSet1 As ResultSet
    Private RowID As Int

    'we check if all fields are filled
    If edtFirstName.Text = "" Or edtLastName.Text = "" Or edtCity.Text = "" Then
        MsgBox("One or more data is missing", "Missing data")
        Return
    End If

    'we check if an entry with the same name already exists
    Query = "SELECT * FROM persons WHERE FirstName = ? AND LastName = ? AND City = ?"
    ResultSet1 = SQL1.ExecQuery2(Query, Array As String(edtFirstName.Text, edtLastName.Text,
    edtCity.Text))

    If ResultSet1.NextRow = True Then
        'if it exists show a message and do nothing else
        ToastMessageShow("This entry already exists", False)
    Else
        'if not, add the entry
        'we use ExecNonQuery2 because it's easier, we don't need to take care of the data types
        Query = "INSERT INTO persons VALUES (?, ?, ?)"
        SQL1.ExecNonQuery2(Query, Array As String(edtFirstName.Text, edtLastName.Text,
        edtCity.Text))

        ToastMessageShow("Entry added", False) ' confirmation message for the user

        'to display the ID of the last entry we read the max value of the internal 'rowid' column
        RowID = SQL1.ExecQuerySingleResult("SELECT max(rowid) FROM persons")
        RowIDList.Add(RowID) 'add the last ID to the list
        CurrentIndex = RowIDList.Size - 1 'set the current index to the last one
        lblRowID.Text = RowID 'display the last index
    End If
    ResultSet1.Close 'close the ResultSet, we don't it anymore
End Sub
```

Small differences for the three operating systems:

- B4A  
 MsgBox("One or more data is missing", "Missing data")  
 ToastMessageShow("This entry already exists", False)
- B4i  
 MsgBox("One or more data is missing", "Missing data")  
 HUD1.ToastMessageShow("Entry added", False) ' confirmation for the user
- B4J the ToastMessageShow function is a routine in the program.  
 fx.Msgbox(MainForm, "One or more data is missing", "Missing data")  
 ToastMessageShow("This entry already exists", False)

### 2.6.2.5 DeleteEntry

We ask the user if he really wants to delete the selected entry.

If the answer is yes, then we delete it.

And set the new CurrentIndex.

```
Sub DeleteEntry
  Private Query As String
  Private Answ As Int

  'ask the user for confirmation
  Answ = MsgBox2("Do you really want to delete " & edtFirstName.Text & " " &
    edtLastName.Text, "Delete entry", "Yes", "", "No", Null)

  If Answ = DialogResult.POSITIVE Then      'if yes, delete the entry
    Query = "DELETE FROM persons WHERE ID = " & IDList.Get(CurrentIndex)
    SQL1.ExecNonQuery(Query)               'delete the entry
    IDList.RemoveAt(CurrentIndex)          'remove the ID from the list
    If CurrentIndex = RowNumber - 1 Then    'if the current index is the last one
      CurrentIndex = CurrentIndex - 1      'decrement it by 1
    End If
    RowNumber = RowNumber - 1              'decrement the row count by 1
    ShowEntry(CurrentIndex)                'show the next entry
    ToastMessageShow("Entry deleted", False) 'confirmation for the user
  End If
End Sub
```

Small differences for the three operating systems:

- B4A / B4J  
  ToastMessageShow("This entry already exists", False)
- B4i  
  HUD1.ToastMessageShow("Entry added", False) ' confirmation for the user

### 2.6.2.6 UpdateEntry

We use ExecNonQuery2 instead of ExecNonQuery because it's easier, we don't need to take care of the data type.

The ? sign is a placeholder for the data which must be given in the array.

```
Sub UpdateEntry
  Private Query As String

  Query = "UPDATE persons Set FirstName = ?, LastName = ?, City = ? _
  WHERE ID = " & IDList.Get(CurrentIndex)
  SQL1.ExecNonQuery2(Query, Array As String(edtFirstName.Text, _
    edtLastName.Text, edtCity.Text))
  ToastMessageShow("Entry updated", False)
End Sub
```

Small differences for the three operating systems:

- B4A / B4J  
  ToastMessageShow("This entry already exists", False)
- B4i  
  HUD1.ToastMessageShow("Entry added", False) ' confirmation for the user

## 2.7 SQLite Database second simple example program SQLiteLight2

This example program is an evolution of the SQLiteLight1 project.

The source codes are located in these folders:

*SQLiteDatabase\_SourceCode\SQLiteLight2\B4A*

*SQLiteDatabase\_SourceCode\SQLiteLight2\B4i*

*SQLiteDatabase\_SourceCode\SQLiteLight2\B4J.*

The program generates a default database if there is none available.

Added a screen displaying the database in a table, using a WebView.

Following functions are implemented:

- Add an entry
- Edit an entry
  - First, Previous, Next and Last entry
  - Update an entry
  - Delete an entry
  - Go to First, Prev, Next and Last entry
- Filter
  - AND / OR Boolean operator
  - Filter

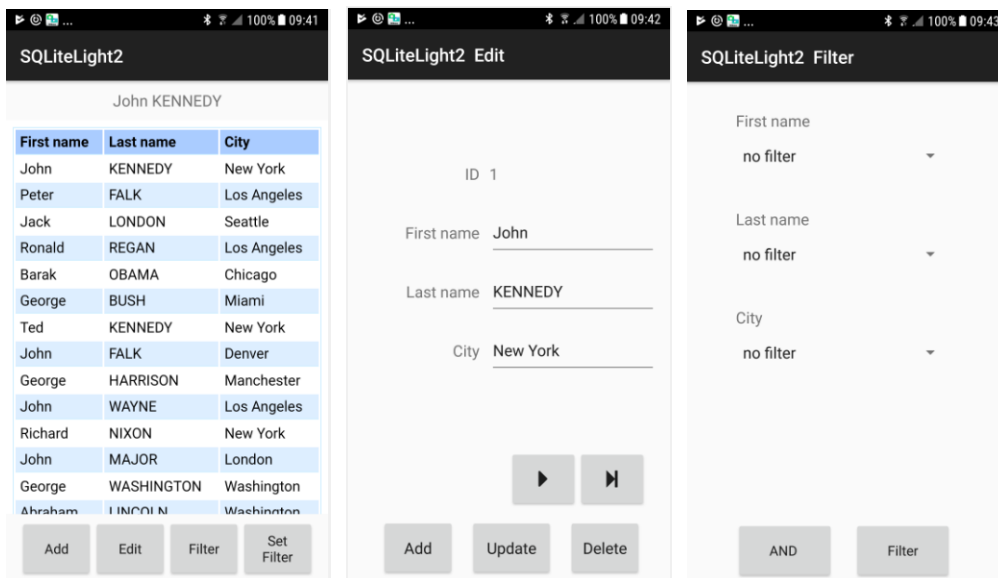
The user interfaces are somewhat different between the three operating systems.

I preferred making the user interfaces more operation system like with specific objects, rather than making them almost the same, especially for B4A and B4i

**This project does also exist as a B4XPages cross-platform project in the B4XPages Cross-platform projects booklet.**

This project is cross-platform oriented with the same interface for all three platforms.

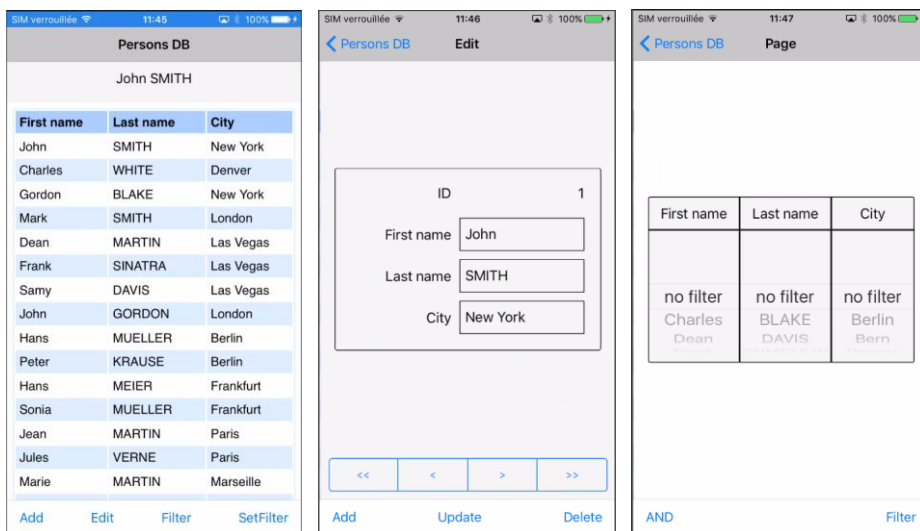
It uses the B4XTable library and XUI Views instead of the WebView and platform specific views.



B4A Main screen

Edit screen

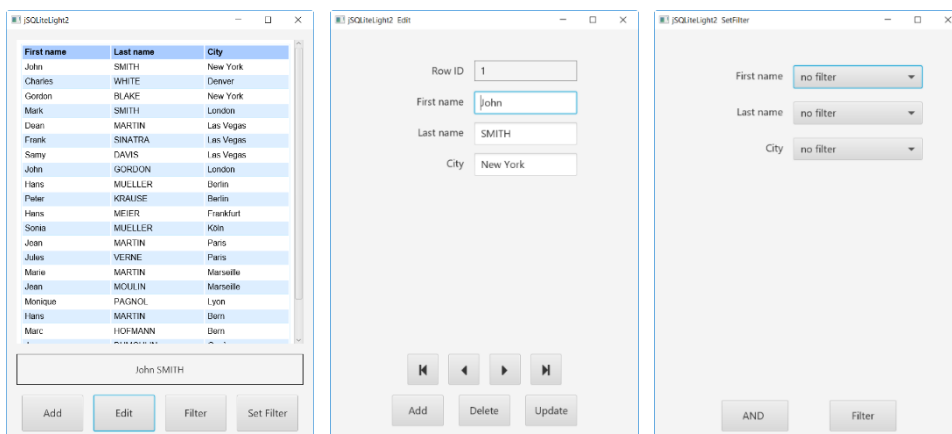
Filter screen



B4i Main screen

Edit screen

Filter screen



B4J Main screen

Edit screen

Filter screen

## 2.7.1 Main module source code parts

Only the SQLite related routines are shown. Operating system routines are not shown.

### 2.7.1.1 Declaration of the Process global variables

We declare operating system dependent variables either in **Process\_Globals** (B4i, B4J) or **Globals** in B4A.

```
Sub Process_Globals
    ' different operating system variables
    '
    '
    'operating system independent variables
    Public SQL1 As SQL

    Public CurrentIndex = -1 As Int    'index of the current entry

    Public RowIDList As List    'list containing the IDs of the database
    'we need it because the IDs can be different from the list indexes
    'if we delete an entry its ID is lost

    ' used in ExecuteHTML
    Private HtmlCSS As String
    HtmlCSS = "table {font-family:helvetica;width: 100%;border: 1px solid #cef;text-align: left; }" _
        & " th { font-weight: bold; background-color: #acf; border-bottom: 1px solid #cef; }" _
        & "td,th {padding: 4px 5px; }" _
        & ".odd {background-color: #def; } .odd td {border-bottom: 1px solid #cef; }" _
        & "a { text-decoration:none; color: #000;}"
End Sub
```

### 2.7.1.2 Show table

We define the SQL query.

Depending if the filter is active, we add the filter query and change the filter button text.

We load the database query result in a WebView and read the database IDs.

```
'Shows the database in a table in a WebView
```

```
Private Sub ShowTable
```

```
    Private Query As String
```

```
    Query = "SELECT FirstName As [First name], LastName As [Last name], City FROM  
persons"
```

```
    'depending if the filter is active or not we add the filter query at the end of the  
query
```

```
    'the filter query is defined in the Filter Activity
```

```
    If Filter.flagFilterActive = False Then
```

```
        btnFilter.Text = "Filter" 'change the text in the Filter button
```

```
    Else
```

```
        Query = Query & Filter.Query
```

```
        btnFilter.Text = "UnFilter" 'change the text in the Filter button
```

```
    End If
```

```
    'displays the database in a table
```

```
    wbvTable.LoadHtml(ExecuteHtml(SQL1, Query, Null, True))
```

```
    ReadDataBaseIDs
```

```
End Sub
```



### 2.7.1.3 ExecuteHtml show a table in a WebView

This routine generates the Html string for the LoadHtml method.

It is extracted from the DBUtils class.

It is the same for all three operating systems.

```
'This routine is extracted from the DBUtils code module
'Creates a html text that displays the data in a table.
'The style of the table can be changed by modifying HtmlCSS variable.
Private Sub ExecuteHtml(SQL As SQL, Query As String, StringArgs() As String, Clickable
As Boolean) As String
    Private ResultSet1 As ResultSet
    If StringArgs <> Null Then
        ResultSet1 = SQL.ExecQuery2(Query, StringArgs)
    Else
        ResultSet1 = SQL.ExecQuery(Query)
    End If
    Private sb As StringBuilder
    sb.Initialize
    sb.Append("<html><body>").Append(CRLF)
    sb.Append("<style type='text/css'>").Append(HtmlCSS).Append("</style>").Append(CRLF)
    sb.Append("<table><tr>").Append(CRLF)
    For i = 0 To ResultSet1.ColumnCount - 1
        sb.Append("<th>").Append(ResultSet1.GetColumnName(i)).Append("</th>")
    Next

    sb.Append("</tr>").Append(CRLF)

    Private row As Int
    row = 0
    Do While ResultSet1.NextRow
        If row Mod 2 = 0 Then
            sb.Append("<tr>")
        Else
            sb.Append("<tr class='odd'>")
        End If
        For i = 0 To ResultSet1.ColumnCount - 1
            sb.Append("<td>")
            If Clickable Then
                sb.Append("<a href='http://'").Append(i).Append(".")
                sb.Append(row)
                sb.Append(".stub'>").Append(ResultSet1.GetString2(i)).Append("</a>")
            Else
                sb.Append(ResultSet1.GetString2(i))
            End If
            sb.Append("</td>")
        Next
        sb.Append("</tr>").Append(CRLF)
        row = row + 1
    Loop
    ResultSet1.Close
    sb.Append("</table></body></html>")
    Return sb.ToString
End Sub
```

### 2.7.1.4 ReadDatabaseRowIDs

We read the rowids from the database. We need this because the entry numbering is not straightforward. If we delete an entry with a given rowid this one is lost to maintain all the other rowids the same.

The routine is the same for all three operating systems.

```
'Reads the database rowids in RowIDList
Private Sub ReadDataBaseRowIDs
    Private ResultSet1 As ResultSet

    If Filter.flagFilterActive = False Then
        ResultSet1 = SQL1.ExecQuery("SELECT rowid FROM persons")
    Else
        ResultSet1 = SQL1.ExecQuery("SELECT rowid FROM persons" & Filter.Query)
    End If

    'We read only the ID column and put them in the IDList
    RowIDList.Initialize                                'initialize the
ID list
    Do While ResultSet1.NextRow
        RowIDList.Add(ResultSet1.GetInt2(0))            'add the rowid's to the RowID list
    Loop
    If RowIDList.Size > 0 Then
        CurrentIndex = 0                                'set the current index
to 0
    Else
        CurrentIndex = -1                                'set the current index
to -1, no selected item
        ToastMessageShow("No items found", False)
    End If
    ResultSet1.Close
    'close the ResultSet, we don't need it anymore
End Sub
```

### 2.7.1.5 UpdateSelectedEntryDisplay

When the user selects an entry, we display it in a Label.

The routine is the same for all three operating systems.

```
Private Sub UpdateSelectedEntryDisplay
    Private Query As String
    Private ResultSet1 As ResultSet

    Query = "SELECT FirstName, LastName, City FROM persons WHERE rowid = " &
RowIDList.Get(CurrentIndex)
    ResultSet1 = SQL1.ExecQuery(Query)
    ResultSet1.NextRow
    lblSelectedItem.Text = ResultSet1.GetString("FirstName") & " " &
ResultSet1.GetString("LastName") & " " & ResultSet1.GetString("City")
    ResultSet1.Close
End Sub
```

### 2.7.1.6 WebView events \_OverrideUrl / \_LocationChanged

We use a WebView event when the user selects an entry.

The content of the routine is the same, only the event name changes in B4J.

#### B4A and B4i:

```
'Routine from the DBUtils demo program
Private Sub wbvTable_OverrideUrl (Url As String) As Boolean
    'parse the row and column numbers from the URL
    Log(Url)
    Private values() As String
    values = Regex.Split("[.]", Url.SubString(7))
    Private row As Int
    row = values(1)
    CurrentIndex = row
    UpdateSelectedItem
    Return True 'Don't try to navigate to this URL
End Sub
```

The URL variable holds the return value from the WebView event.

It could look like this http//2.7.stub/ where 2 is the col index and 7 is the row index.

The col and row values are extracted in values = Regex.Split("[.]", Url.SubString(7))

values(0) holds the col value

values(1) holds the row value

values(2) holds the end of the string

**B4J:** the OverrideURL event doesn't exist in B4J so we use LocationChanged.

```
Private Sub wbvTable_LocationChanged (Location As String)
    'parse the row and column numbers from the Location string

    Private values() As String
    values = Regex.Split("[.]", Location.SubString(7))

    Private row As Int
    row = values(1)
    CurrentIndex = row
    UpdateSelectedItem
End Sub
```

The Location string holds the return value from the WebView event.

## 2.7.2 Edit Module source code parts

In the Edit module, there is nothing special.

## 2.7.3 Filter Module source code parts

Most of the code is self-explanatory.

For the filter data selection, we use for:

- B4A Spinners
- B4i Pickers
- B4J ComboBoxes

These are filled with the data from the database.

The first item in each object is “no filter”, which means that this column is not filtered.

But as there can be multiple entries with the same data we fill them with ‘distinct’ data, one name is shown only once.

The code is shown for one object only, the principle is the same for the others.

### 2.7.3.1 B4A

```
'Initialize the Spinners
Private Sub UpdateFilters
    Private Query1 As String
    Private ResultSet1 As ResultSet

    'We execute a query for each column and fill the Spinner
    'We use SELECT DISTINCT to have each existing first name in the database only once
    Query1 = "SELECT DISTINCT FirstName FROM persons ORDER BY FirstName ASC"

    'fill the FirstName Spinner
    ResultSet1 = Main.SQLite1.ExecQuery(Query1)
    'we add 'no filter' as no selection
    spnFirstName.Add("no filter")
    'we fill the Spinner with the data from the database
    Do While ResultSet1.NextRow
        spnFirstName.Add(resultSet1.GetString("FirstName"))
    Loop
```

### 2.7.3.2 B4i

We use Pickers.

```
'Initialize the Pickers
Private Sub UpdateFilters
    Private Query1 As String
    Private ResultSet1 As ResultSet

    'We execute a query for each column and fill the Spinner
    'We use SELECT DISTINCT to have each existing first name in the database only once

    'fill FirstName Picker
    Query1 = "SELECT DISTINCT FirstName FROM persons ORDER BY FirstName ASC"

    Private lst As List
    lst.Initialize

    'we add 'no filter' as no selection
    lst.Add("no filter")
    ResultSet1 = Main.SQLite1.ExecQuery(Query1)

    'we fill the Picker with the data from the database
    Do While ResultSet1.NextRow
        lst.Add(resultSet1.GetString("FirstName"))
    Loop
    picFirstName.SetItems(0, lst)
```

### 2.7.3.3 B4J

We use ComboBoxes.

```
'Initialize the ComboBoxes
Private Sub UpdateFilters
    Private Query1 As String
    Private ResultSet1 As ResultSet

    'We execute a query for each column and fill the ComboBox
    'We use SELECT DISTINCT to have each existing first name in the database only once
    Query1 = "SELECT DISTINCT FirstName FROM persons ORDER BY FirstName ASC"

    'fill the FirstName ComboBox
    ResultSet1 = Main.SQLite1.ExecQuery(Query1)
    'we add 'no filter' as no selection
    cbxFirstName.Items.Clear
    cbxFirstName.Items.Add("no filter")
    'we fill the Spinner with the data from the database
    Do While ResultSet1.NextRow
        cbxFirstName.Items.Add(resultSet1.GetString("FirstName"))
    Loop
```

## 2.8 SQLite Database third simple example program SQLiteLight3

A third example program is in the SQLiteLight3 program.

The source codes are located in these folders:

*SQLiteDatabase\_SourceCode\SQLiteLight3\B4A*

*SQLiteDatabase\_SourceCode\SQLiteLight3\B4i*

*SQLiteDatabase\_SourceCode\SQLiteLight3\B4J.*

This program is almost the same as SQLiteLight2, all functions are the same.

The differences are the database path, database name, table name, column number, column names, column alias names and column data types are variables instead being hard coded.

It allows also to generate a new database by:

- changing in Globals the values of the variables listed above
- in Activity\_Create
- comment this line: File.Copy(File.DirAssets, SQLDateBaseName, SQLDataBasePath, SQLDateBaseName)
- uncomment this line: CreateDataBase

The code has comments and is, I hope, self explanatory.

One example to show the difference:

For the query to show the table.

In SQLiteLight2 the names are hard coded:

```
Sub ShowTable
  Private i As Int
  Private Query As String
  Query = "SELECT FirstName As [First name], LastName As [Last name], _
          City FROM persons"
```

In SQLiteLight3 the names are variables defined in Globals:

```
Sub ShowTable
  Private i As Int
  Private Query As String

  Query = "SELECT "
  For i = 0 To ColNumber - 1
    If i < ColNumber - 1 Then
      Query = Query & ColNames(i) & " As [" & ColAliasNames(i) & "], "
    Else
      Query = Query & ColNames(i) & " As [" & ColAliasNames(i) & " ] "
    End If
  Next
  Query = Query & " FROM " & SQLTableName
```

## 2.9 SQLite Database 3<sup>rd</sup> example with B4XTable

This project is the same as the previous one, but it uses B4XTable to display the database instead of a WebView.

## 2.10 SQLite Database 3<sup>rd</sup> example XUI version SQLiteLight3X

This program is still the third SQLite Database example project, but a XUI cross platform version.

Most of the code is in common modules!

The project uses:

- A common Starter module.  
The Starter service module is the best entry point for B4A. To share most of the code between the three platforms we use also the same Starter module in B4i and B4J.
- A B4XTable table to display the data.
- XUI Views for the interfaces.



## 2.11 SQLite Database fourth example program SQLiteLight4

This SQLite example program, SQLiteLight4, is a bit more elaborated than SQLiteLight2.

In SQLiteLight2 there is only one table, in this program there are three tables.

The purpose of this example is to show the principle of managing several tables.

To make the code more readable, all names are hard coded and not stored in variables like in SQLiteLight3.

The source codes are located in these folders:

*SQLiteDatabase\_SourceCode\SQLiteLight4\B4A*

*SQLiteDatabase\_SourceCode\SQLiteLight4\B4i*

*SQLiteDatabase\_SourceCode\SQLiteLight4\B4J.*

The program manages a spare part stock. The tables are intentionally very simple with just a few columns and not all possible errors or mistakes a user can make are checked to keep the code simple and easier to read and understand.

The database has three tables:

- |             |                                       |                                     |                                   |
|-------------|---------------------------------------|-------------------------------------|-----------------------------------|
| • Stock     | Number INTEGER,<br>number of products | ProductID INTEGER,<br>product ID    | Date INTEGER<br>date in Ticks     |
| • Products  | Name TEXT,<br>product name            | Price REAL,<br>product price        | SupplierID INTEGER<br>supplier ID |
| • Suppliers | Name TEXT,<br>suppliers' name         | Address TEXT,<br>suppliers' address | City TEXT<br>suppliers' city      |

In the table Stock we use the ID of the product rather its name. The same in the table Products for the Supplier. The advantage is that we memorize a reference to the data in the original table instead of copying the data into another table. If we change once the data in the original table all the data in other tables are updated automatically.

Query example of a call for display:

```
Query = "SELECT Stock.Number, Products.Name AS Product, Suppliers.Name AS Supplier,
Products.Price AS Price, Stock.Number * Products.Price AS Value, date(Stock.Date /
1000, 'unixepoch') AS Date"
```

```
Query = Query & " FROM Stock, Products, Suppliers"
```

```
Query = Query & " WHERE Products.rowid = Stock.ProductID AND Suppliers.rowid =
Products.SupplierID"
```

We want to read following data:

- The number of items in stock `Stock.Number`.  
The *Number* column in the *Stock* table.
- The product name `Products.Name AS Product`.  
The *Name* column in the *Products* table and give this column the name 'Product'.
- The supplier name `Suppliers.Name AS Supplier`.  
The *Name* column in the *Suppliers* table and give this column the name 'Supplier'.
- The product price `Products.Price AS Price`.  
The *Price* column in the *Products* table and give this column the name 'Price'.
- The value of these products in stock `Stock.Number * Products.Price AS Value`.  
The multiplication of the number of items in stock with the product price and give this column the name 'Value'.
- The date when the product was entered `date(Stock.Date / 1000, 'unixepoch') AS Date`.  
We use the SQLite date function where we give the *Date* column of the *Stock* table.  
As the date is in B4A ticks we need to divide the value by 1000 to adapt it to 'SQL ticks' and we must add the parameter 'unixepoch' for 'SQL ticks'.

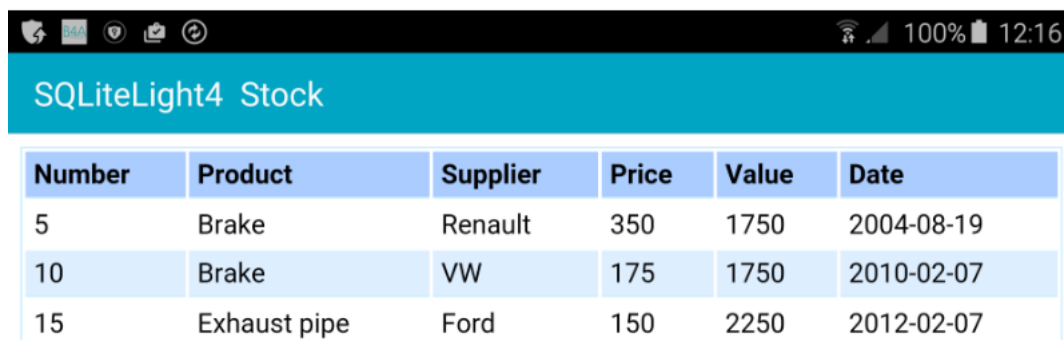
The query concerns the three tables Stock, Products and Suppliers:

`FROM Stock, Products, Suppliers`

We must add a WHERE expression:

- To connect the *Products* table *rowid* to the *Stock ProductID* column value.  
`Products.rowid = Stock.ProductID`
- To connect the *Suppliers* table *rowid* to the *Products SupplierID* column value.  
`Suppliers.rowid = Products.SupplierID`

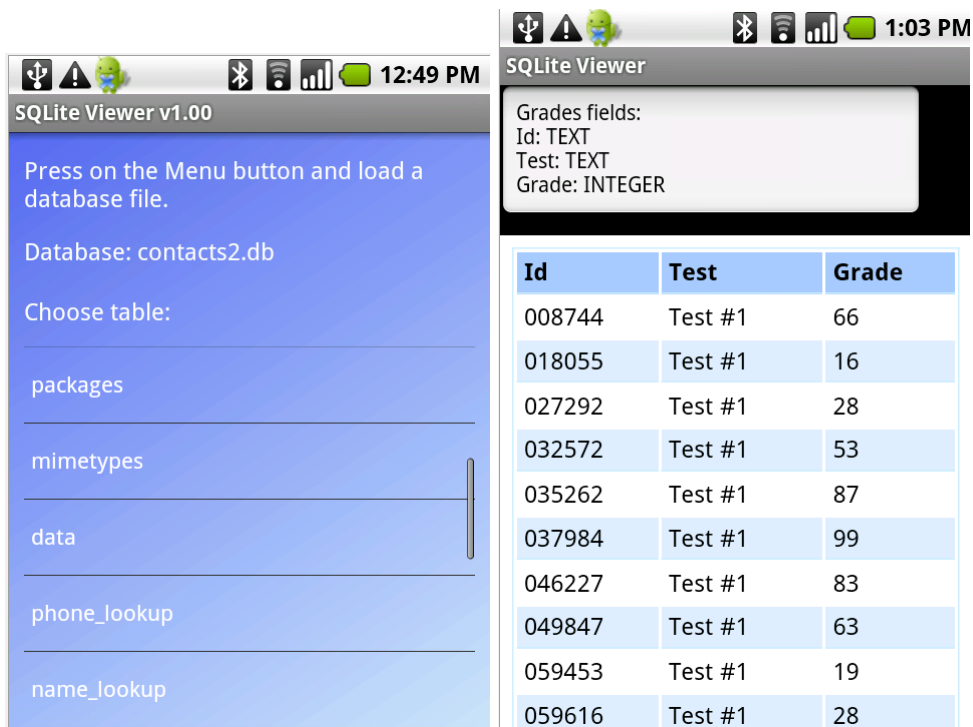
Example of the result:



Number	Product	Supplier	Price	Value	Date
5	Brake	Renault	350	1750	2004-08-19
10	Brake	VW	175	1750	2010-02-07
15	Exhaust pipe	Ford	150	2250	2012-02-07

## 2.12 SQLite Viewer

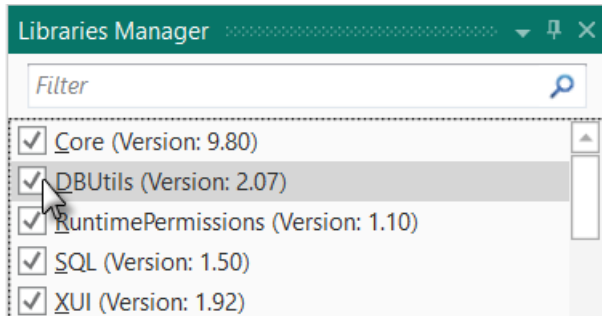
There is a B4A [SQLiteViewer](#) program in the forum, that allows you to load and display databases. The program uses the [DBUtils](#) module and the table is shown in a WebView view. The usage of the DBUtils module is explained in the [DBUtils chapter](#).



### 3 DBUtils version 2








For those who are not familiar with SQLite, Erel has written DBUtils, a B4X Library, that should make things easier.

To use it, check it in the Libraries Manager Tab.



If you don't have it yet, download it from the forum and save the DBUtils.b4xlib file into the \AdditionalLibraries\B4X folder.

If have not yet defined the AdditionalLibraries folder with the structure below, it's time to do it.

- ▼  AdditionalLibraries
  -  B4A Folder for B4A additional libraries.
  -  B4i Folder for B4i additional libraries.
  -  B4J Folder for B4J additional libraries.
  - >  B4R Folder for B4R additional libraries.
  -  B4X Folder for B4X libraries.
  -  B4XlibXMLFiles Folder for B4X libraries XML files.

One subfolder for each product: B4A, B4i, B4J, B4R and another B4X for B4X libraries.

B4X Libraries are explained in chapter [B4X libraries x.b4xlib](#) in the B4X Basic Language booklet.

### 3.1 DBUtil functions

Following functions are available:

Methods	B4A	B4i	B4J
CopyDBFromAssets	x	x	x
CreateTable	x	x	x
DeleteRecord	x	x	x
DropTable	x	x	x
ExecuteHtml	x	x	x
ExecuteJSON	x	x	x
ExecuteList	x	x	x
ExecuteList2	x	x	x
ExecuteListView	x		
ExecuteMap	x	x	x
ExecuteMemoryTable	x	x	x
ExecuteTableView			x
ExecuteSpinner	x		
GetDBFolder	x	x	x
GetDBVersion	x	x	x
GetFieldsInfo	x	x	x
GetTables	x	x	x
InsertMaps	x	x	x
SetDBVersion	x	x	x
TableExists	x	x	x
UpdateRecord	x	x	x
UpdateRecord2	x	x	x

### 3.1.1 CopyDBFormAssets B4A, B4

**CopyDBFromAssets**(FileName As String) As String

Copies a database file that was added in the project Files tab. The database must be copied to a writable location because it is not possible to access a database located in File.DirAssets.

This method copies the database in:

- **B4A**, to the storage card with `rp.GetSafeDirDefaultExternal`. If the storage card is not available, the file is copied to the internal folder `File.DirInternal`.
- **B4i**, to `File.DirDocuments`.

The target folder is returned. If the database file already exists, then no copying is done.

### 3.1.2 CopyDBFormAssets B4J

**CopyDBFromAssets** (FileName As String, AppName As String) As String

Copies a database file that was added in the project Files tab. The database must be copied to a writable location because it is not possible to access a database located in File.DirAssets.

This method copies the database in:

- **B4J**, to `C:\Users\UserName\AppData\Roaming\AppName`  
UserName your user name  
AppName the given App name

The target folder is returned. If the database file already exists, then no copying is done.

### 3.1.3 CreateTable B4A, B4i, B4J

**CreateTable**(SQL As SQL, TableName As String, FieldsAndTypes As Map, PrimaryKey As String)

Creates a new table with the given name.

FieldsAndTypes - A map with the fields names as keys and the types as values.

You can use the DB\_... constants for the types.

PrimaryKey - The column that will be the primary key. Pass empty string if not needed.

### 3.1.4 DeleteRecord B4A, B4i, B4J

**DeleteRecord**(SQL As SQL, TableName As String, WhereFieldEquals As Map)

Deletes records.

WhereFieldEquals, Map with the WHERE conditions.

Map Key = Column,

Map Value = WHERE condition value

### 3.1.5 DropTable B4A, B4i, B4J

**DropTable**(SQL As SQL, TableName As String)  
Deletes the given table.

### 3.1.6 ExecuteHtml B4A, B4i, B4J

**ExecuteHtml**(SQL As SQL, Query As String, StringArgs() As String, Limit As Int, Clickable As Boolean) As String

Creates an html text that displays the data in a table in a WebView.

The style of the table can be changed by modifying the HtmlCSS variable.

StringArgs() - Values to replace question marks in the query. Pass Null if not needed.

Limit - Limits the results. Pass 0 for all results.

### 3.1.7 ExecuteJSON B4A, B4i, B4J

**ExecuteJSON**(SQL As SQL, Query As String, StringArgs() As String, Limit As Int, DBTypes As List) As Map

Executes the given query and creates a Map that you can pass to JSONGenerator and generate JSON text.

StringArgs()- Values to replace question marks in the query. Pass Null if not needed.

Limit - Limits the results. Pass 0 for all results.

DBTypes - Lists the type of each column in the result set.

Usage example: (don't forget to add a reference to the JSON library)

```
Dim gen As JSONGenerator
gen.Initialize(DBUtils.ExecuteJSON(SQL, "SELECT Id, Birthday FROM Students",
Null, 0, Array As String(DBUtils.DB_TEXT, DBUtils.DB_INTEGER)))
Dim JSONString As String
JSONString = gen.ToPrettyString(4)
Msgbox(JSONString, "")
```

### 3.1.8 ExecuteList B4A, B4i, B4J

**ExecuteList**(SQL As SQL, Query As String, StringArgs() As String, Limit As Int, List1 As List)

Executes the query and fills the List with the values.

StringArgs()- Values to replace question marks in the query. Pass Null if not needed.

Limit - Limits the results. Pass 0 for all results.

### 3.1.9 ExecuteListView B4A

**ExecuteListView**(SQL As SQL, Query As String, StringArgs() As String, Limit As Int, ListView1 As ListView, TwoLines As Boolean)

Executes the query and fills the ListView with the values.

StringArgs()- Values to replace question marks in the query. Pass Null if not needed.

Limit - Limits the results. Pass 0 for all results.

If TwoLines is true then the first column is mapped to the first line and the second column is mapped to the second line.

In both cases the value set to the row is the array with all the records values.

### 3.1.10 ExecuteMap B4A, B4i, B4J

**ExecuteMap**(SQL As SQL, Query As String, StringArgs() As String) As Map

Executes the query and returns a Map with the column names as the keys and the first record values As the entries values.

StringArgs() - Values to replace question marks in the query. Pass Null if not needed.

The keys are lower cased.

Returns Null if no results found.

### 3.1.11 ExecuteMemoryTable B4A, B4i, B4J

**ExecuteMemoryTable**(SQL As SQL, Query As String, StringArgs() As String, Limit As Int) As List

Executes the query and returns the result as a list of arrays.

Each item in the list is a strings array.

StringArgs() - Values to replace question marks in the query. Pass Null if not needed.

Limit - Limits the results. Pass 0 for all results.

### 3.1.12 ExecuteTableView B4J

**ExecuteTableView**(SQL As SQL, Query As String, StringArgs() As String, Limit As Int, \_ TableView1 As TableView)

Executes the query and fills the TableView with the values.

StringArgs()- Values to replace question marks in the query. Pass Null if not needed.

Limit - Limits the results. Pass 0 for all results.

### 3.1.13 ExecuteSpinner B4A

**ExecuteSpinner**(SQL As SQL, Query As String, StringArgs() As String, Limit As Int, Spinner1 As Spinner)

Executes the query and fills the Spinner with the values.

StringArgs()- Values to replace question marks in the query. Pass Null if not needed.

Limit - Limits the results. Pass 0 for all results.



### 3.1.14 GetDBFolder B4A, B4i, B4J

**GetDBFolder** *As String*

Returns the path to a folder where you can create a database, preferably on the secondary storage.

### 3.1.15 GetDBVersion B4A, B4i, B4J

**GetDBVersion**(SQL *As SQL*) *As Int*

Gets the current version of the database.

If the DBVersion table does not exist it is created and the current version is set to version 1.

### 3.1.16 GetFieldInfo B4A, B4i, B4J

**GetFieldInfo**(SQL *As SQL*, TableName *As String*)

Gets information about each field in a table.

Returns a list of DBFieldInfo

### 3.1.17 GetTables B4A, B4i, B4J

**GetTables**(SQL *As SQL*)

Get all tables names as list.

Returns: List

### 3.1.18 InsertMaps B4A, B4i, B4J

**InsertMaps**(SQL *As SQL*, TableName *As String*, ListOfMaps *As List*)

Inserts the data to the table.

ListOfMaps - A list with maps as items. Each map represents a record where the map keys are the columns names and the maps values are the values. Note that you should create a new map for each record (this can be done by calling Dim to redim the map).

### 3.1.19 SetDBVersion B4A, B4i, B4J

**SetDBVersion**(SQL *As SQL*, Version *As Int*)

Sets the database version to the given version number.

### 3.1.20 TableExists B4A, B4i, B4J

**TableExists**(SQL *As SQL*, TableName *As String*) *As Boolean*

Tests whether the given table exists.

### 3.1.21 UpdateRecord B4A, B4i, B4J

**UpdateRecord**(SQL As SQL, TableName As String, Field As String, NewValue As Object, WhereFieldEquals As Map)

Updates a single record in the database.

Field - Column name

NewValue - new value

WhereFieldEquals - Map where the map keys are the column names and the map values the values to update.

### 3.1.22 UpdateRecord2 B4A, B4i, B4J

**UpdateRecord2**(SQL As SQL, TableName As String, Fields As Map, WhereFieldEquals As Map)

Updates multiple records in the database.

Fields – Map where the map keys are the column names and the map values the new value.

WhereFieldEquals - Map where the map keys are the column names and the map values the values to update.

## 3.2 Examples

You find Erels' examples in the Forum under: [\[B4X\] DBUtils 2](#).

These examples are not explained in this chapter.

### 3.3 DBUtilsDemo example program

This example program shows the use of some DBUtils features.

The database used is personsflca.db, which contains persons data:

- FirstName
- LastName
- Address
- City

The source codes are located in these folders:

*SQLiteDatabase\_SourceCode\DBUtilsDemo\B4A*  
*SQLiteDatabase\_SourceCode\DBUtilsDemo \B4i*  
*SQLiteDatabase\_SourceCode\DBUtilsDemo \B4J.*

They need following libraries:

- SQL
- XUI

Most of the code is the same for all three products B4A, B4i and B4J.

DBUtils functions used:

- DBUtils.CopyDBFromAssets
- DBUtils.ExecuteHTML
- DBUtils.ExecuteMemoryTable
- DBUtils.ExecuteList
- DBUtils.UpdateRecors2
- DBUtils.InsertMaps
- DBUtils.DeleteRecord

The code is not explained in detail, I think that it is enough self-explanatory.

jDBUtilsDemo

First name  Last name

Address  City

First name	Name	Address	City
John	SMITH	Broadway 505	New York
Charles	WHITE	Oak Lane 1245	Denver
Gordon	BLAKE	Broadway 1505	New York
Mark	SMITH	Christopher St. 234	London
Dean	MARTIN	W. Harmon Ave. 1200	Las Vegas
Frank	SINATRA	W. Russel Rd. 123	Las Vegas
Samy	DAVIS	E. Flamingo Rd. 347	Las Vegas
John	GORDON	Bonhill St. 12	London
Hans	MUELLER	Bahnhofstr. 134	Berlin
Peter	KRAUSE	Kurfürstendamm 201	Berlin
Hans	MEIER	Baslerstr. 14	Frankfurt
Sonia	MUELLER	Kronbergerstr. 56	Frankfurt
Jean	MARTIN	Champs Elisées 243	Paris
Jules	VERNE	Rue Saint-Honoré 25	Paris
Marie	MARTIN	Rue du Port 23	Marseille

Add Update Delete

B4J

DBUtilsDemo

First name John

Last name SMITH

Address Broadway 505

City New York

First name	Last Name	Address	City
John	SMITH	Broadway 505	New York
Charles	WHITE	Oak Lane 1245	Denver
Gordon	BLAKE	Broadway 1505	New York
Mark	SMITH	Christopher St. 234	London
Dean	MARTIN	W. Harmon Ave. 1200	Las Vegas
Frank	SINATRA	W. Russel Rd. 123	Las Vegas
Samy	DAVIS	E. Flamingo Rd. 347	Las Vegas
John	GORDON	Bonhill St. 12	London
Hans	MUELLER	Bahnhofstr. 134	Berlin

Add Update Delete

B4A

iPhone de... 15:47 95%

iDBUtilsDemo

First name John

Last name SMITH

Address Broadway 505

City New York

First name	Last Name	Address	City
John	SMITH	Broadway 505	New York
Charles	WHITE	Oak Lane 1245	Denver
Gordon	BLAKE	Broadway 1505	New York
Mark	SMITH	Christopher St. 234	London
Dean	MARTIN	W. Harmon Ave. 1200	Las Vegas
Frank	SINATRA	W. Russel Rd. 123	Las Vegas
Samy	DAVIS	E. Flamingo Rd.	Las

Add Update Delete

B4i

### 3.3.1 Code differences

The main code differences between the three products are the start of the program which are operating system specific and the WebView event routines.

The WebView event routines are the same in B4A and B4i but different in B4J.

**B4A and B4i**, the event is called **OverrideUrl**.

```
Private Sub WebView1_OverrideUrl (Url As String) As Boolean
    'parse the row and column numbers from the URL
    Private values() As String
    values = Regex.Split("[.]", Url.SubString(7))
    SelectedCol = values(0)
    SelectedRow = values(1)

    UpdateSelectedData    'updates the selected entry

    Return True 'Don't try to navigate to this URL
End Sub
```

**B4J** the event is called **LocationChanged**.

```
Private Sub WebView1_LocationChanged (Location As String)
    'parse the row and column numbers from the URL
    Private values() As String
    If Location.Contains("stub") Then
        values = Regex.Split("[.]", Location.SubString(7))
        SelectedCol = values(0)
        SelectedRow = values(1)

        UpdateSelectedData    'updates the selected entry
    End If
End Sub
```