

NATIONAL CYBER CRIME CONFERENCE

APRIL 26-27, 2021

MSAB LAB: SHINE THE LITE ON SQLITE FORENSICS

Chris Carrier, Technical Trainer

LAB:

SpotliteSqlite <http://github.com/XamnR/SQLite>

RESOURCES:

SQLite Organization <http://sqlite.org>

TOOLS:

HxD <http://mh-nexus.de/en/hxd/>

SQLite Expert www.sqliteexpert.com/

Database Browser <http://sqlitebrowser.org/dl/>

James Eichbaum's Tempus, Base 64 Decoder, and SQLParser <http://github.com/eichbaumj/>
(/Python and /SQLite-Database-Analyzer)

Python www.python.org



MSAB's Advanced Application Analysis Training

www.msab.com/training/

MSAB's XRY, XAMN Spotlight, Horizon & Elements

www.msab.com/products/

1.1.1 Introduction

SQLite Databases

- The Associated Headers
- Deleted Data
- Variable Integers (VARINTS)
- Binary Large Objects (Base64 Encoding)

Python and SQLite

1.1.2 SQLite Data We Want

- Contact Information
- Groups
- Messages
- Date/Time Stamps
- Location
- Attachment Information
- Binary Large Objects: Large files such as: Apple Property Lists, XML, and pictures
- Deleted Data

1.1.3 Epoch Time Stamps

Mac HFS+ Time Stamp

Start Date: January 1, 1904

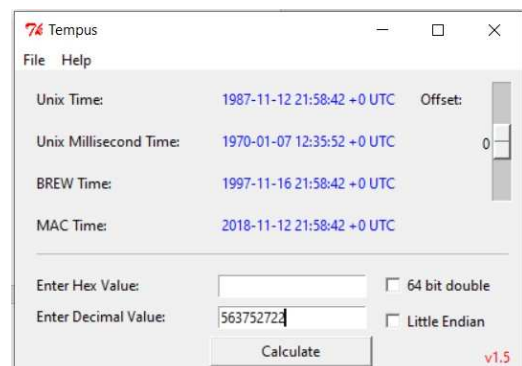
Example: 3624904694

www.epochconverter.com/mac

Mac Absolute Time

Start Date: January 1, 2001

Example: 563752722

The image shows a screenshot of a software application window titled "Tempus". The window has a menu bar with "File" and "Help". Below the menu bar, there are four rows of time conversion data: "Unix Time:" with the value "1987-11-12 21:58:42 +0 UTC", "Unix Millisecond Time:" with "1970-01-07 12:35:52 +0 UTC", "BREW Time:" with "1997-11-16 21:58:42 +0 UTC", and "MAC Time:" with "2018-11-12 21:58:42 +0 UTC". To the right of these values is a vertical slider labeled "Offset:" with a value of "0". Below this, there are two input fields: "Enter Hex Value:" and "Enter Decimal Value:". The "Enter Decimal Value:" field contains the number "563752722". To the right of these fields are two checkboxes: "64 bit double" and "Little Endian". Below the input fields is a "Calculate" button. In the bottom right corner of the window, the version number "v1.5" is displayed.

For Mac Time stamps ignore the period and everything to the right of the period. i.e.

509653685.733396 www.epochconverter.com/coredata

Unix

Start Date: January 1, 1970

Example: 1542059734

10 Digits = Seconds

13 Digits = Milliseconds

www.epochconverter.com/

Offline Tool Tempus: <http://github.com/eichbaumj/Python>

1.1.4 Databases

Databases are used to store larger amounts of data in an orderly and efficient fashion

You will find databases in everything from huge data centers like TAX Revenue organizations, call centers and even your cellphone

Normally they contain a big file that holds the data, and a configuration (config) file that tells the computer where to find the data

1.1.5 Smart Phone SQLite Databases

Compact

Cross platform

Reliable

Zero configuration

- (all in one file)

1.1.6 SQLite Database Data Types

Data types found in databases include strings, integers, reals, BLOBs, JSONs, and NULLs. You may also find subsets of certain data types stored as VarInts.

Data types can store textual based data such as ASCII and Unicode, but also encoded data such as base64 encoded data.

Strings	The easiest of the data types to interpret. The data is usually already in readable text. Strings are normally used for ASCII, Unicode, or Base64 encoded data
Integers	Whole numbers of a certain length. Integers are useful for timestamps and are good for storing other number sets which have constant lengths
Reals	Real numbers provide for the storage of numbers with decimal precision. Real numbers are the same as floats when compared to programming languages.
BLOB	Binary Large Object – often represent large data such as pictures, audio, or video. BLOBs can also be used to hide data in a database

Strings are the easiest type to interpret, meaning the data is (somewhat) readable. Strings are normally used for ASCII, Unicode, or Base64 encoded text.

Integers are whole numbers of a certain length. They are very useful for timestamps and might be a good way to store any other known length number, like phone numbers. They are also used as primary keys. A primary key is a unique number that grows as each new record is added to a table. The primary key is optional, but most databases will make use of it.

There is another type of integer that is designed for variable lengths. This type is called a VarInt.

Real numbers are the same as floating point numbers. A floating point value is a decimal value that is stored in an 8 byte sequence.

BLOB's stands for Binary Large Objects and often represent larger piece of data, like pictures, sound or video clips. They can also be used to hide data in a database. Several pieces of data can be stored together into one BLOB that only the app can understand.

A JSON is an object that can hold several attributes. Just as the objects in python can hold many values of an object (like a car object can have attributes of make, model and year) a JSON can hold many values in a database cell. JSON's are meant to be easily read, making them easy to interpret.

NULL means nothing. NULL should not be confused with zero. Zero is a value where NULL has no value whatsoever.

All data stored in databases are categorized when they are stored and are stored as a long line of data. For example, a row in a database of a very basic chat app may contain 6 columns; a ROW id, the other correspondent's phone number, the correspondent's name or ID number, the actual text, a time stamp and a record ID. The text is categorized as strings, the numbers as integers or floats, and so on.

When we look at the content through a database viewer, the viewer will sort it all out for us and present it in a readable format. Data is usually easy to interpret when viewed through a viewer, but some textual data can be stored using Base64 encoding, making it difficult to decipher.

1.1.7 Building and Modifying a SQLite Database

To gain a better understanding of what happens when a database is created and when records are added to and deleted from tables, we will manually create a database using the different data types associated with SQLite databases.

To get a good foundation, we will create the database via command line rather than using a database program that creates everything behind the scenes for us.

The files you will need for this lab if you want to participate with your computer can be downloaded from <http://github.com/Xamnr/SQLite>

Find a place to put the Sqlite Folder on your computer. Open Windows Explorer and navigate to the C:\ drive where you put the folder and open it. There are three files: sqlite3.def; sqlite3.dll; and sqlite3.exe

CTRL + SHIFT and then right click the sqlite3 folder. Select 'Open command window here' from the right click context menu. The command prompt window will appear:



SQLite is a free package, consisting of just three files. There is no installation process. All three files need to be in the same folder. Sqlite3.exe must be run from a DOS window. Learning the complete ins and outs of databases creation and manipulation would be another course all by itself. For our purposes, we want to build a basic database just to see how it is structured and where data is placed within the file.

To create a new database, type the following and press Enter:

sqlite3 company.db

This will create a new database in memory called, "company.db". We will store employee information for a fictitious company in this database. The command prompt has changed from C:/sqlite3> to just sqlite> with a flashing cursor waiting for your next command.

Right now the database is empty. There are no tables. To exit the database, type the following and press Enter:

.quit

You are now back at the Dos command prompt within the sqlite3 folder. When checking the directory, the database is not there. The database was not saved because it was completely empty.

1.1.8 Creating Tables

This time we will create a table to store our data. Earlier you learned tables are made up of fields, or columns. Those fields identify certain data types that can be placed in them. For this example, we need fields to contain the following information about our employees:

ID	First Name	Last Name	Date of Hire	Age	Gender	Title
Integer	String	String	Datetime	Integer	Integer	String

Open the database once again using the following command and press Enter:

sqlite3 company.db

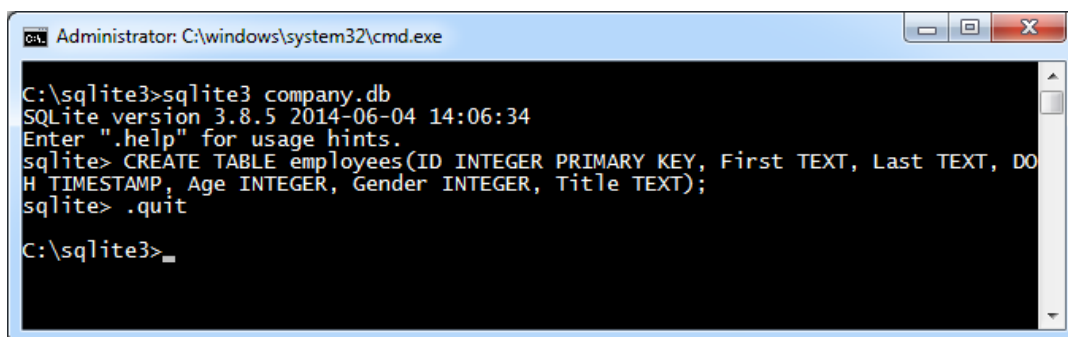
Type the following at the command prompt and press Enter to create a table called employees:

CREATE TABLE employees(ID INTEGER PRIMARY KEY, First TEXT, Last TEXT, DOH TIMESTAMP, Age INTEGER, Gender INTEGER, Title TEXT);

Type the following to exit the database:

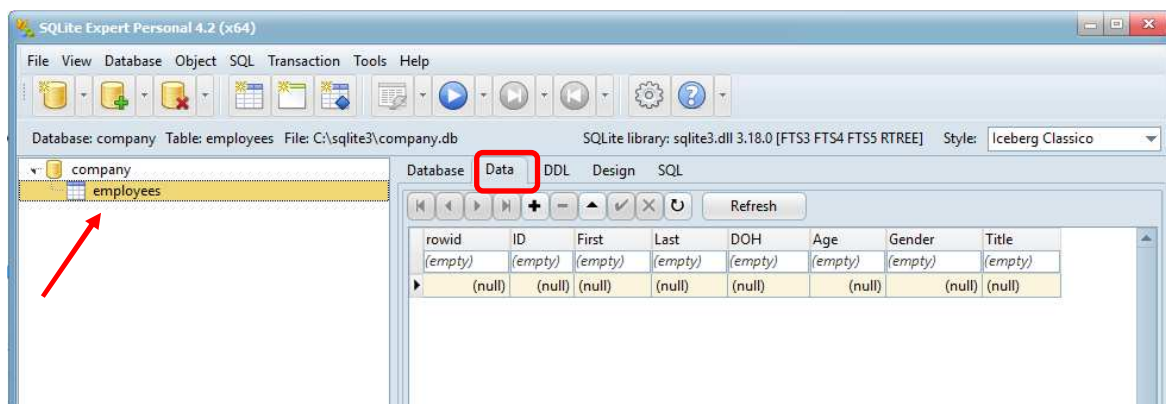
.quit

The sequence of events should appear as they do in the following screenshot:

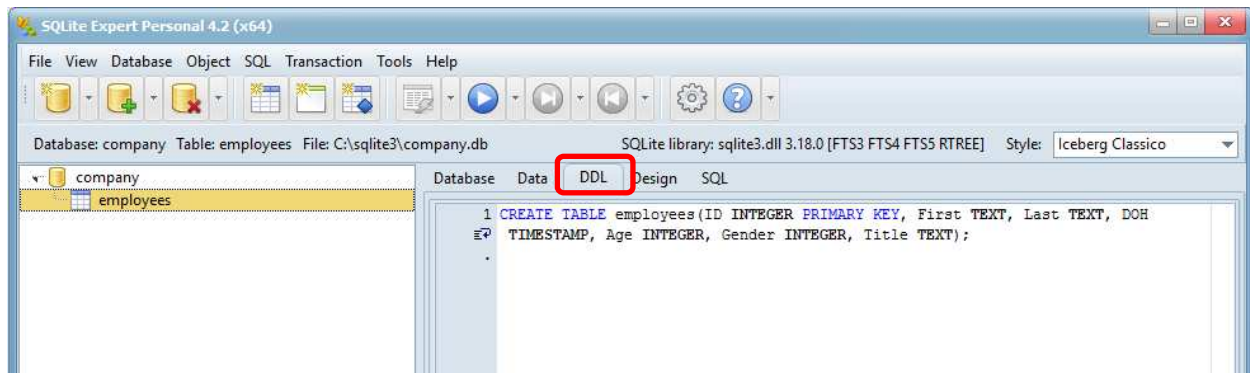


```
C:\sqlite3>sqlite3 company.db
SQLite version 3.8.5 2014-06-04 14:06:34
Enter ".help" for usage hints.
sqlite> CREATE TABLE employees(ID INTEGER PRIMARY KEY, First TEXT, Last TEXT, DOH TIMESTAMP, Age INTEGER, Gender INTEGER, Title TEXT);
sqlite> .quit
C:\sqlite3>
```

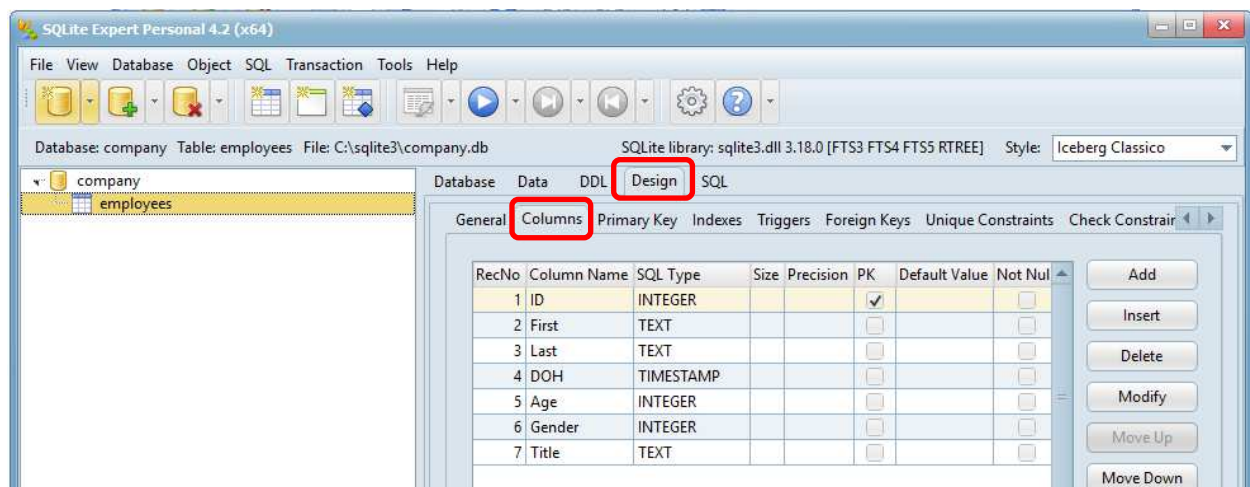
Open 'company.db' in SQLite Expert. Select the employees table and be sure the Data tab is selected in the right window. All of the fields created with the SQLite CREATE statement appear as column headers.



The DDL tab in the right window contains the CREATE TABLE statement from our database:



The Design tab in the right window lists all of the data types for each field in the table:



Later we will examine the database headers to get information such as page size and the number of pages within the database. But for now, this database is currently 2,048 bytes in size. There are two pages within the database. Each page is 1,024 bytes in size. Page 2 currently holds no records, whereas page one contains 1 record.

Close SQLite Expert and open 'company.db' in HxD. The main database header consists of 100 bytes. Immediately following that header is the header for page 1. The header informs us that there is 1 record within this page and it can be found at offset 0x0361 or decimal 865 from the start of the page.

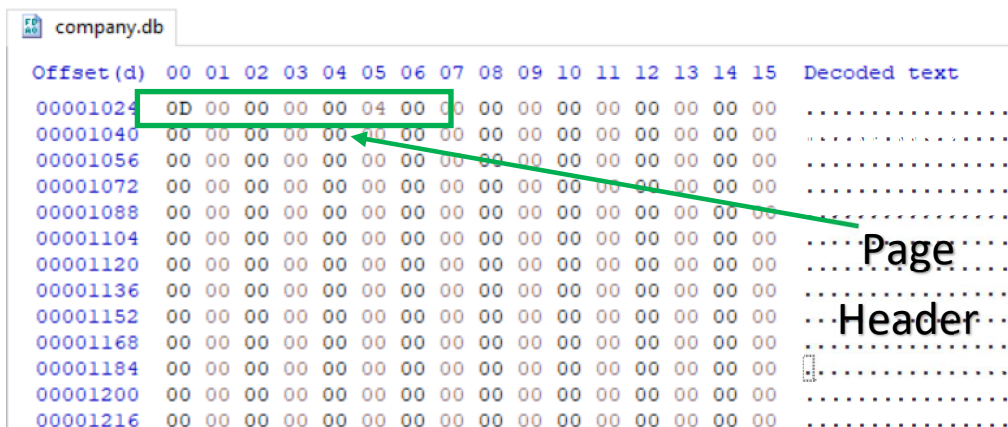
Offset (d)	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	Decoded text
00000000	53	51	4C	69	74	65	20	66	6F	72	6D	61	74	20	33	00	SQLite format 3.
00000016	04	00	01	01	00	40	20	20	00	00	00	01	00	00	00	02@
00000032	00	00	00	00	00	00	00	00	00	00	00	01	00	00	00	04
00000048	00	00	00	00	00	00	00	00	00	00	00	01	00	00	00	00
00000064	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	01
00000096	00	2D	E6	05	0D	00	00	00	01	03	61	00	03	61	00	00	..æ.....a..a..
00000112	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000128	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000704	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000720	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000736	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000752	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000768	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000784	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000800	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000816	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000832	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000848	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000864	00	81	1C	01	07	17	1F	1F	01	82	07	74	61	62	6C	65,table
00000880	65	6D	70	6C	6F	79	65	65	73	65	6D	70	6C	6F	79	65	employeesemploye
00000896	65	73	02	43	52	45	41	54	45	20	54	41	42	4C	45	20	es.CREATE TABLE
00000912	65	6D	70	6C	6F	79	65	65	73	28	49	44	20	49	4E	54	employees(ID INT
00000928	45	47	45	52	20	50	52	49	4D	41	52	59	20	4B	45	59	EGER PRIMARY KEY
00000944	2C	20	46	69	72	73	74	20	54	45	58	54	2C	20	4C	61	, First TEXT, La
00000960	73	74	20	54	45	58	54	2C	20	44	4F	48	20	54	49	4D	st TEXT, DOH TIM
00000976	45	53	54	41	4D	50	2C	20	41	67	65	20	49	4E	54	45	ESTAMP, Age INTE
00000992	47	45	52	2C	20	47	65	6E	64	65	72	20	49	4E	54	45	GER, Gender INTE
00001008	47	45	52	2C	20	54	69	74	6C	65	20	54	45	58	54	29	GER, Title TEXT)

Going to offset 865 takes us to the start of the record. This record is the Create Table statement for our employees table. It is the only record in the page and is located at the bottom of the page. But what page can we find any records for this table? You can see it pointed out here:

00000864	00	81	1C	01	07	17	1F	1F	01	82	07	74	61	62	6C	65,table
00000880	65	6D	70	6C	6F	79	65	65	73	65	6D	70	6C	6F	79	65	employeesemploye
00000896	65	73	02	43	52	45	41	54	45	20	54	41	42	4C	45	20	es.CREATE TABLE
00000912	65	6D	70	6C	6F	79	65	65	73	28	49	44	20	49	4E	54	employees(ID INT
00000928	45	47	45	52	20	50	52	49	4D	41	52	59	20	4B	45	59	EGER PRIMARY KEY
00000944	2C	20	46	69	72	73	74	20	54	45	58	54	2C	20	4C	61	, First TEXT, La
00000960	73	74	20	54	45	58	54	2C	20	44	4F	48	20	54	49	4D	st TEXT, DOH TIM
00000976	45	53	54	41	4D	50	2C	20	41	67	65	20	49	4E	54	45	ESTAMP, Age INTE
00000992	47	45	52	2C	20	47	65	6E	64	65	72	20	49	4E	54	45	GER, Gender INTE
00001008	47	45	52	2C	20	54	69	74	6C	65	20	54	45	58	54	29	GER, Title TEXT)
00001024	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

This will be covered in detail later. But for now, records for this table can currently be found on page 2 within the database.

Page 2 starts at offset 1,024. Scroll through the entire page and notice that it is completely empty, except for the page header. As records for the employees table are added to the database, page two will be populated with those records, starting from the bottom of the page and working their way up.



Offset(d)	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	Decoded text
00001024	0D	00	00	00	00	04	00	00	00	00	00	00	00	00	00	00
00001040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001056	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001072	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001088	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001104	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001120	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001136	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001152	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001168	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001184	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001200	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001216	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Close HxD. Right click 'company.db' and click copy. Paste a copy of the database to the Desktop. Add copy to the end of company. So, it is now companycopy.db We will use the copy later to analyze changes made to the database.

1.1.9 Inserting Records

The database is ready to accept records. These records could come from an app running on an iPhone, an Android, or a Windows Phone. The app could prompt the user to enter all of the details required to complete each field within the table and could also be updated by the user. Let's go ahead and manually add a record to this database. This will represent an app adding data through user interaction with their phone.

Go back to the command prompt and type the following and press Enter to reopen the database for editing:

sqlite3 company.db

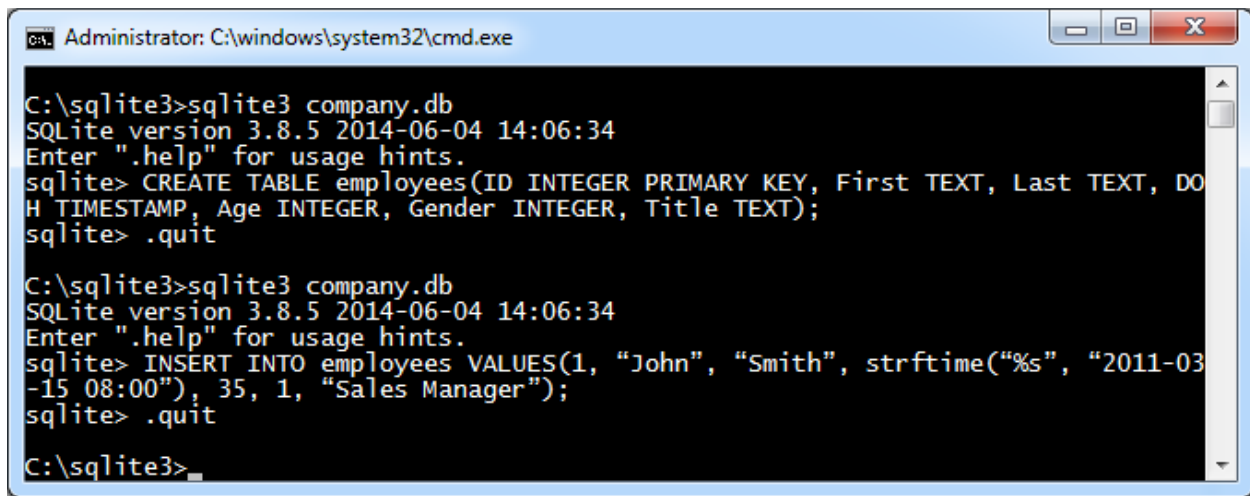
Type the following and press Enter to create a new record with the given details:

INSERT INTO employees VALUES(1, "John", "Smith", strftime("%s", "2011-03-15 08:00"), 35, 1, "Sales Manager");

Type the following and press Enter to close the database:

.quit

So far the commands we have entered look like this:



```
C:\sqlite3>sqlite3 company.db
SQLite version 3.8.5 2014-06-04 14:06:34
Enter ".help" for usage hints.
sqlite> CREATE TABLE employees(ID INTEGER PRIMARY KEY, First TEXT, Last TEXT, DOH
H TIMESTAMP, Age INTEGER, Gender INTEGER, Title TEXT);
sqlite> .quit

C:\sqlite3>sqlite3 company.db
SQLite version 3.8.5 2014-06-04 14:06:34
Enter ".help" for usage hints.
sqlite> INSERT INTO employees VALUES(1, "John", "Smith", strftime("%s", "2011-03
-15 08:00"), 35, 1, "Sales Manager");
sqlite> .quit

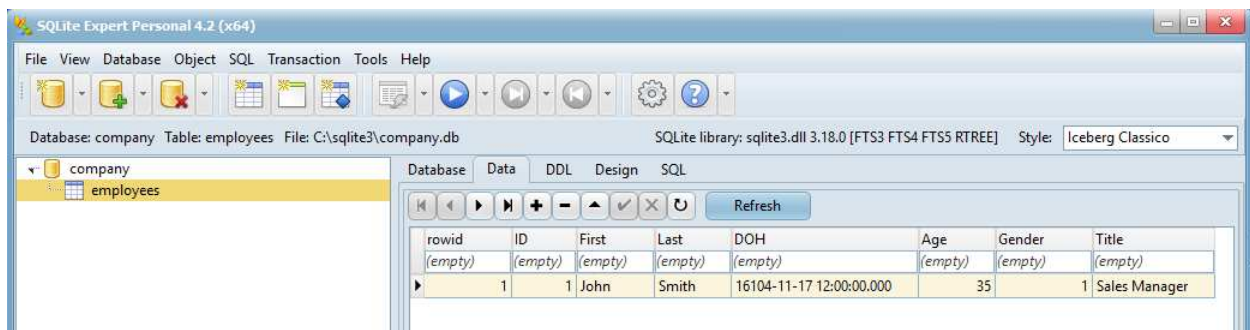
C:\sqlite3>
```

The SQLite insert statement describes exactly what is happening. You are inserting a record into the employees table. The record contains the values described. String values are enclosed with quotes. Integer values do not have quotes. The timestamp is provided as a string and is converted to a time format to be stored by the database.

A new record in the employees table has been created and should have the following information:

ID	First Name	Last Name	Date of Hire	Age	Gender	Title
1	John	Smith	2011-03-15 08:00	35	1	Sales Manager

Open 'company.db' in SQLite Expert and view the record within the employees table.



You will notice the timestamp under the DOH column does not look at all how we expect it to look. SQLite Expert 3 is not displaying it properly. SQLite Expert Personal 4 will display time in this format and others, as you have seen before with UNIX timestamps displayed in their decimal representation. You can also view the record details from the command line within sqlite3. Close SQLite Expert.

Go back to the command window and type the following, pressing Enter after each line:

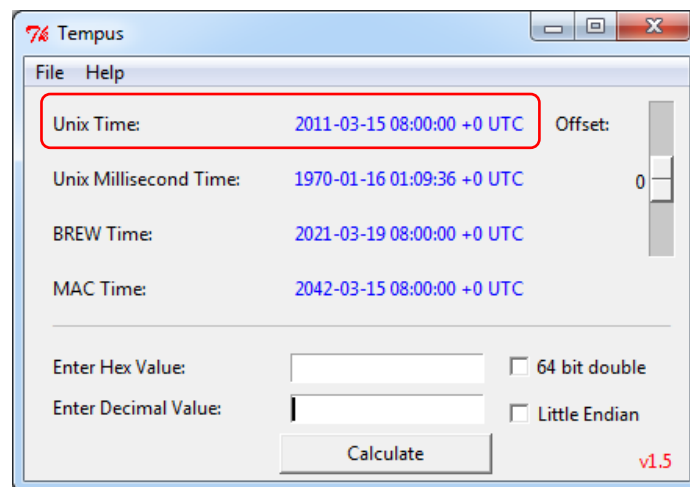
```
sqlite3 company.db
```

```
select * from employees;
```

SQLite3 responds to the select command by displaying the following to the screen:

```
1|John|Smith|1300176000|35|1|Sales Manager
```

From here, we can see that the timestamp is a UNIX timestamp and can be converted using a tool such as Tempus. The converted value in Tempus is:

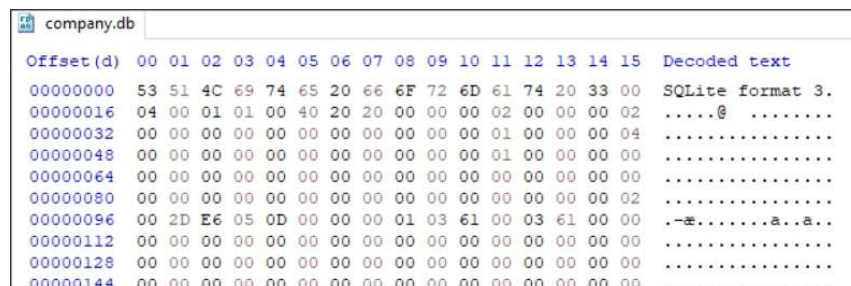


Type the following to exit the database:

```
.quit
```

Analyzing the SQLite Database Header

1. Open the **company.db** in HxD (or you can use the figure below). In HxD click on View and select Data Inspector.



Offset(d)	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	Decoded text
00000000	53	51	4C	69	74	65	20	66	6F	72	6D	61	74	20	33	00	SQLite format 3.
00000016	04	00	01	01	00	40	20	20	00	00	00	02	00	00	00	02@
00000032	00	00	00	00	00	00	00	00	00	00	00	01	00	00	00	04
00000048	00	00	00	00	00	00	00	00	00	00	00	01	00	00	00	00
00000064	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	02
00000096	00	2D	E6	05	0D	00	00	00	01	03	61	00	03	61	00	00	..a.....a..
00000112	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000128	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000144	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

2. What is the page file size? _____
3. How many pages does the database contain? _____

4. What is the size of the database in bytes? _____
5. How many freelist pages does the database contain? _____
6. What is the page number of the first freelist trunk page? _____
7. What version of SQLite was used to create the database? _____

SQLite Database Header (first 100 bytes of the first page)

OFFSET	SIZE	DESCRIPTION	ADDITIONAL INFORMATION
0	16	Header string: "SQLite format 3"	
16	2	Page Size – Big Endian	10 00 = 4096 bytes per page
18	1	File Format Write Version 1 = Legacy / 2 = WAL	
19	1	File Format Read Version 01 = Legacy / 22 = WAL	01 = Legacy (roll back journal) 02 = Write Ahead Log (WAL) and Shared Memory File (SHM)
20	1	Unused space at end of each page	00 = none
24	4	File Change Counter	
28	4	Size of database in pages (3.7.0+)	(Pages * 4096 = Size of Database in Bytes)
32	4	Page # of first freelist trunk page	
36	4	Total # of freelist pages	
48	4	Default Cache Size	
52	4	Page # of Largest root b-tree page	
56	4	Database Text Encoding	00 00 00 01 = 1 (UTF-8 or ASCII)
60	4	The "User Version"	
64	4	Vacuum mode	00 00 00 01 = 1 (Incremental-Vacuum Mode)
68	24	Reserved. Must be 00	All 00's
92	4	Version-Valid-Number	(Matches Offset 24)
96	4	SQLite Version Number	00 2D E2 28 = 3007016

This table and the other tables may be found at sqlite.org

Analyzing John Smith Record

1. Locate the John Smith record you created in the database. Consider using CTRL-F and searching for the text string "John"

Where is the record found? _____

2. The time field came right after the last name. Knowing the date and time is stored in UNIX time format, can you manually decode the timestamp from the hex values found after "Smith"?

3. Go to offset 1,024 of the most current version (company.db) of the database.

Offset 1,024 is the beginning of page 2.

Address	Hex	ASCII
	00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15	
.....1024	0D 00 00 00 01 03 DB 00 03 DB 00 00 00 00 00 00	0 0
.....1040	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	

The pointer in this array is pointing to a record at offset 0x03DB. This is the offset from the start of the page.

4. Use Windows Calculator in programmer mode (or HxD's Data inspector) to convert 0x03DB to decimal. What is the converted value?

5. Starting from the beginning of page 2, offset 1,024, highlight the number of bytes you converted from 0x03DB from the start of page 2. Where does this place the cursor?

6. Return to the command prompt window. Type the following and press Enter to open the database for editing:

sqlite3 company.db

Add one more record to the database with the following command and press Enter:

INSERT INTO employees VALUES(2, "Jane", "Doe", strftime("%s", "2014-12-01 15:00"), 44, 0, "Marketing Director");

Exit the database by typing the following and press Enter:

.quit

7. In HxD view the company.db and scroll to the bottom.

Where was the new record placed? _____

8. Go to offset 1,024 of the latest version of the database. Has the pointer array to the records been altered? If so, how?

As new records are added to the database, a page fills up from the bottom upwards. The top of the page contains the header and a pointer array. As new records are added, pointers to those records are also added to the array. The records and the array will eventually converge upon each other. When this happens, a new page will be created to store new records.

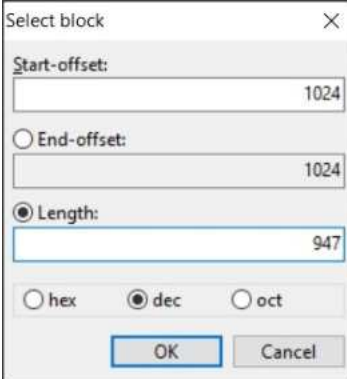


Offset (d)	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	Decoded text
00001024	0D	00	00	00	02	03	B3	00	03	DB	03	B3	00	00	00	00Û.ª.....
00001040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001056	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001072	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001088	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001104	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001120	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001136	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001152	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001168	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

03B3 converted from hex to decimal (Big Endian) is 947 as shown above in HxD Data inspector view.

Put the cursor at the beginning of page two. In HxD click Edit and select block. Select Length and type in 947 bytes. Make sure dec (decimal is selected).

This should highlight the freespace from the Record pointer to the start of the last record inserted.



Select block

Start-offset: 1024

End-offset: 1024

Length: 947

hex dec oct

OK Cancel

1.1.10 Delete Cell/Record

Let's delete the first record we entered into the database. We assigned the value 1 to the ID field for that record. Open the database for editing by typing the following and then press Enter:

Sqlite3 company.db

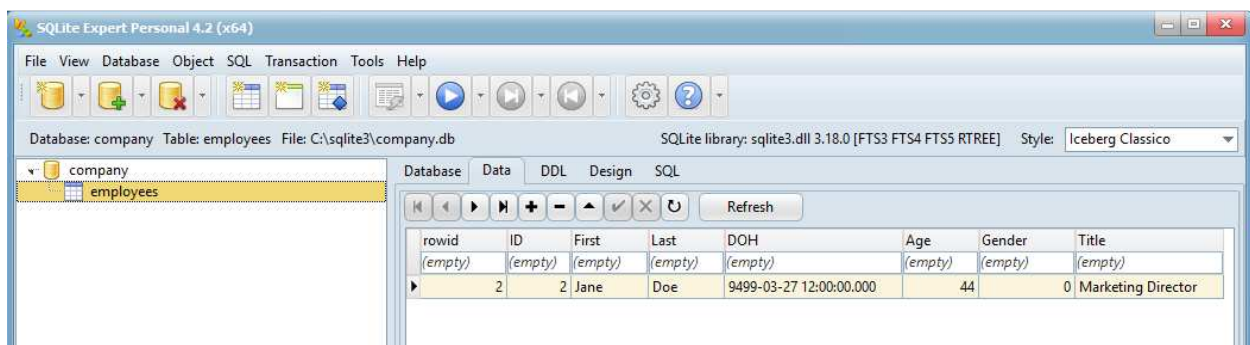
Delete the record where the value for ID is 1 by typing the following and press Enter:

DELETE FROM employees WHERE ID = 1;

Exit the database by typing the following and then press Enter:

.quit

We can see that the record has been deleted by opening 'company.db' in SQLite Expert and viewing the only live record in the employee table:



SQLite Expert Personal 4.2 (x64)

File View Database Object SQL Transaction Tools Help

Database: company Table: employees File: C:\sqlite3\company.db SQLite library: sqlite3.dll 3.18.0 [FTS3 FTS4 FTS5 RTREE] Style: Iceberg Classico

rowid	ID	First	Last	DOH	Age	Gender	Title
(empty)	(empty)	(empty)	(empty)	(empty)	(empty)	(empty)	(empty)
2	2	Jane	Doe	9499-03-27 12:00:00.000	44	0	Marketing Director

As you can see, "John Smith" has been deleted from this database.

But when this new company.db is viewed in HxD, we can see that the record is still in the same location:

```

-----
00001776 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00001792 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00001808 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00001824 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00001840 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00001856 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00001872 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00001888 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00001904 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00001920 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00001936 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00001952 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00001968 00 00 00 26 02 08 00 15 13 04 01 08 31 4A 61 6E ...&.....lJan
00001984 65 44 6F 65 54 7C 82 70 2C 4D 61 72 6B 65 74 69 eDoeT|,p,Marketi
00002000 6E 67 20 44 69 72 65 63 74 6F 72 00 00 00 25 15 ng Director...%.
00002016 17 04 01 09 27 4A 6F 68 6E 53 6D 69 74 68 4D 7F ....'JohnSmithM.
00002032 1C 80 23 53 61 6C 65 73 20 4D 61 6E 61 67 65 72 .€#Sales Manager

```

The pointer array, however, no longer has an entry pointing to the record we deleted. The pointer array only points to record number 2:

company.db

Offset (d)	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	Decoded text
00001024	0D	03	DB	00	01	03	B3	00	03	B3	00	B3	00	00	00	00	..Û...³...³....
00001040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001056	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001072	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001088	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

1 Record Offset to first record

What happens when a new record is added? Does it now overwrite the very first and already deleted record? Let's find out. Close HxD. Open the 'company.db' file by typing the following and press Enter:

sqlite3 company.db

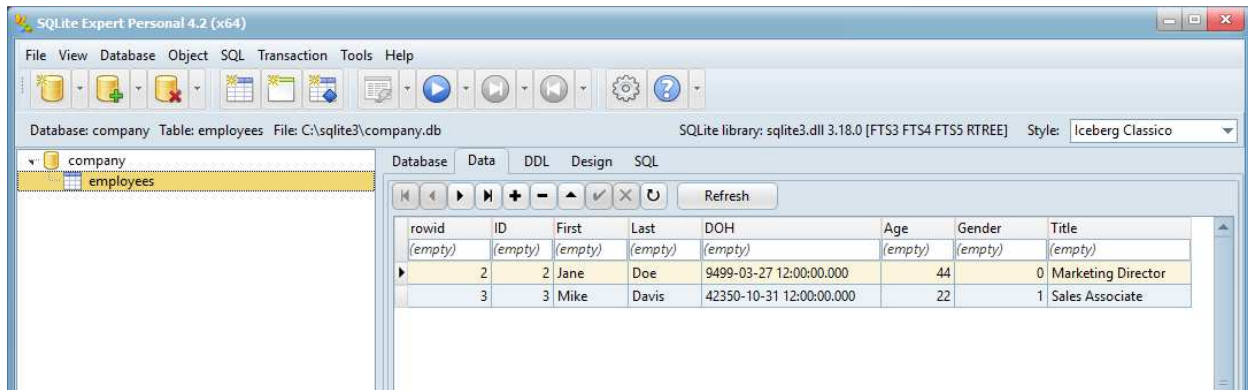
Add a new record to the employees table by typing the following and press Enter:

INSERT INTO employees VALUES(3, "Mike", "Davis", strftime("%s", "2015-04-19 12:00"), 22, 1, "Sales Associate");

Close the database by typing the following and press Enter:

.quit

Viewing the employees table of the database in SQLite Expert displays the two live records:



Opening 'company.db' in HxD will show if the record for "John Smith" still remains in its original location within the file.

1. Is John Smith still present?

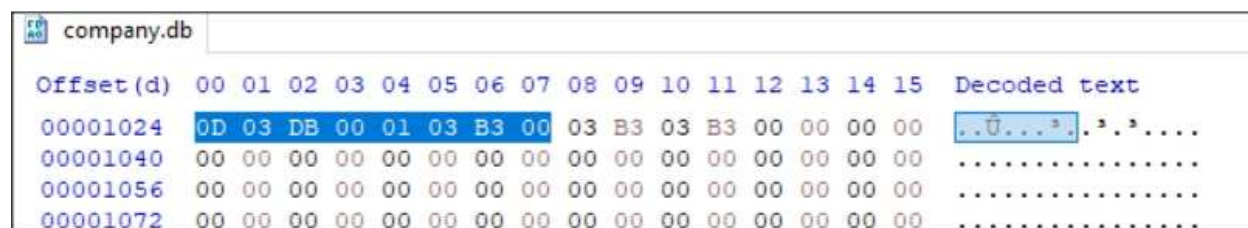
Offset (d)	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	Decoded text
00001664	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001680	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001696	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001712	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001728	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001744	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001760	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001776	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001792	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001808	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001824	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001840	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001856	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001872	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001888	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001904	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001920	00	00	00	00	00	00	00	00	00	00	00	00	25	03	08	00%
00001936	15	17	04	01	09	2B	4D	69	6B	65	44	61	76	69	73	55+MikeDavisU
00001952	33	98	C0	16	53	61	6C	65	73	20	41	73	73	6F	63	69	3~A.Sales Associ
00001968	61	74	65	26	02	08	00	15	13	04	01	08	31	4A	61	6E	ate&.....lJan
00001984	65	44	6F	65	54	7C	82	70	2C	4D	61	72	6B	65	74	69	eDoeT ,p,Marketi
00002000	6E	67	20	44	69	72	65	63	74	6F	72	00	00	00	25	15	ng Director...%.
00002016	17	04	01	09	27	4A	6F	68	6E	53	6D	69	74	68	4D	7F'JohnSmithM.
00002032	1C	80	23	53	61	6C	65	73	20	4D	61	6E	61	67	65	72	.€#Sales Manager

1.1.11 Page Headers

The next step in dissecting the database is understanding individual pages. Immediately following the database header is the first page within the database and the page header.

Page Header Analysis

company.db after John Smith was deleted



Offset(d)	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	Decoded text
00001024	0D	03	DB	00	01	03	B3	00	03	B3	03	B3	00	00	00	00	..Ü...
00001040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001056	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001072	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Page one will always have a 100 byte main header for the database. All consecutive pages will be the full default size as indicated in the database header. The following table shows what each page header will consist of.

Page Header Table

OFFSET	SIZE	DESCRIPTION
0	1	B-Tree Flag Type 0x02 = Interior B-Tree Page 0x05 = Interior Table B-Tree Page 0x0A = Leaf Index B-Tree Page 0x0D = Leaf Table B-Tree Page
1	2	Byte offset into the page of the first freeblock
3	2	Number of Cells on this Page
5	2	Offset to the first byte of cell content
7	1	Number of fragmented free bytes in cells
8	4	Right most pointer (Interior b-tree pages only 0x05, 12 byte header)

Interior b-tree pages have a 12 byte header. Leaf pages have an 8 byte header (There will be no 'Right most pointer' entry). Following the header is the pointer array or arrays consisting of 2-byte values pointing to offsets for records within the page.

1. How many cells/records are on this page?_____
2. What is the byte offset into the page of the first freeblock (deleted cell/record) converted to decimal?

3. What is the offset to the first byte of cell content (converted to decimal)?

1.1.12 Freeblock

Offset 1 for two bytes in the page header tells us where the first freeblock is within the page. Freeblocks are deleted records. A freeblock can consist of several concurrent deleted records. Let's use the company database we created on Day 1 as an example. We created three records in the employee table and deleted the very first record. We saw that the record still remained within the database even though we deleted it. We saw it when we viewed that page within HxD. But let's take a closer look at the page header (the current company.db may also be found in the Day 2 sample files).

2nd Page Header after John Smith Deleted

Offset (d)	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	Decoded text
00001024	0D	03	DB	00	01	03	B3	00	03	B3	03	B3	00	00	00	00	...Û...³.³....
00001040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001056	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001072	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

John Smith Start of the Record Header before deleted

00001968	00	00	00	26	02	08	00	15	13	04	01	08	31	4A	61	6E	...&.....lJan
00001984	65	44	6F	65	54	7C	82	70	2C	4D	61	72	6B	65	74	69	eDoeT ,p,Marketi
00002000	6E	67	20	44	69	72	65	63	74	6F	72	23	01	08	00	15	ng Director#....
00002016	17	04	01	09	27	4A	6F	68	6E	53	6D	69	74	68	4D	7F'JohnSmithM.
00002032	1C	80	23	53	61	6C	65	73	20	4D	61	6E	61	67	65	72	.€#Sales Manager

John Smith Start of the Record Header after deletion

Offset (d)	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	Decoded text
00001920	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001936	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001952	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001968	00	00	00	26	02	08	00	15	13	04	01	08	31	4A	61	6E	...&.....lJan
00001984	65	44	6F	65	54	7C	82	70	2C	4D	61	72	6B	65	74	69	eDoeT ,p,Marketi
00002000	6E	67	20	44	69	72	65	63	74	6F	72	00	00	00	25	15	ng Director...§.
00002016	17	04	01	09	27	4A	6F	68	6E	53	6D	69	74	68	4D	7F'JohnSmithM.
00002032	1C	80	23	53	61	6C	65	73	20	4D	61	6E	61	67	65	72	.€#Sales Manager

The page header indicates that the first freeblock can be found at offset 0x03DB or decimal 987. At offset 987 we will find the start of the very first record in the table, the one we deleted. The first four bytes of a deleted record are replaced with two (2) byte values.

The first two byte value will point to the next freeblock (deleted record). The second two byte value indicates the size of the freeblock. In this case, the first two byte value is 0x0000. 0x0000 tells us that there are not more freeblocks. If the value was non-zero, then we could follow the chain to other freeblocks and other deleted records. The second two byte value is 0x00025, or decimal 37. The total length of this freeblock is 37 bytes, including the four byte freeblock header.

If all the records of an entire page are deleted, then the entire page will be marked as a freelist page. None of the record headers will be replaced with the four byte freeblock headers. This makes it nice as the records are easier to recovery in their entirety.

1.1.13 Freelist Pages

The header contains very useful information, including the location of the first freelist trunk page, if there is one.

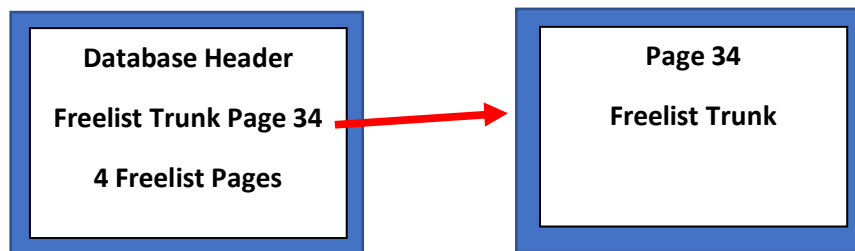
Talk.sqlite Database Header

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0123456789ABCDEF
00	53	51	4C	69	74	65	20	66	6F	72	6D	61	74	20	33	00	SQLite format 3.
16	10	00	01	01	00	40	20	20	00	00	01	1A	00	00	00	25@.....%
32	00	00	00	22	00	00	00	04	00	00	00	1C	00	00	00	01	...".....
48	00	00	00	00	00	00	00	1D	00	00	00	01	00	00	00	00
64	00	00	00	01	00	00	00	00	00	00	00	00	00	00	00	00
80	00	00	00	00	00	00	00	00	00	00	00	00	00	00	01	1A
96	00	2D	E2	28	05	00	00	00	01	0F	FB	00	00	00	00	20	..-.(.....
12	0F	FB	0D	28	0D	B0	0C	03	0B	8B	0B	1C	0A	AD	0A	4E	...(.N
28	09	EF	08	DE	08	8D	08	2D	07	6C	07	F8	06	9F	07	37- .1.....7
44	05	3F	04	E6	04	87	04	22	03	AC	02	52	01	A4	0D	E5	.?....."....R....

Remember a SQLite database header is made up of 100 bytes. The header describes the database layout such as how many pages it contains, how big the pages are, and how many free list pages can be found.

The following table describes each element within the SQLite header:

OFFSET	SIZE	DESCRIPTION	TALK.SQLITE
0	16	Header string: "SQLite format 3"	53 51 4C 69 74 65 20 66 6F 72 6D 61 74 20 33 00
16	2	Page Size – Big Endian	10 00 = 4096 bytes per page
28	4	Size of database in pages (3.7.0+)	00 00 00 25 = 37 (File size 37 * 4096 = 151,552)
32	4	Page # of first freelist trunk page	00 00 00 22 = page 34
36	4	Total # of freelist pages	00 00 00 04 = 4 pages



Let's get right to what we are after. The main database header indicates that the first freelist trunk page was 34. This was found at offset 32 for 4 bytes within the SQLite header. The value was 0x00000022.

This is stored as a big endian integer and converts to decimal 34. To find the offset to this page, use the following formula: **(Page -1) * Pagefile Size = Offset** $(34 - 1) * 4096 = 135,168$

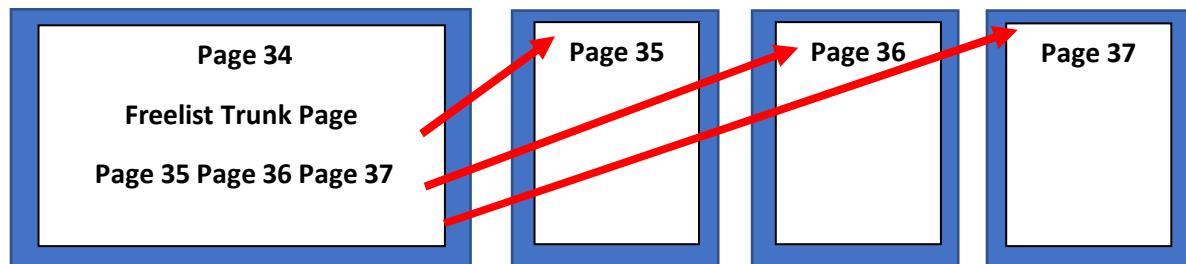
Page 34 Freelist Trunk Page at offset 135,168

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0123456789ABCDEF
00135136	28	5A	5F	33	49	4E	56	49	54	45	44	47	52	4F	55	50	(Z_3INVITEDGROUP
00135152	53	2C	20	5A	5F	37	49	4E	56	49	54	45	45	29	20	29	S, Z_7INVITEE))
00135168	00	00	00	00	00	00	00	03	00	00	00	25	00	00	00	24%...\$
00135184	00	00	00	23	CF	FC	B6	63	5C	58	99	2A	72	8B	E5	4F	...#...c\X.*r..O
00135200	7D	1A	B9	C9	EC	79	A9	49	6D	B6	C7	CB	23	C7	BF	F0	}....y.Im...#...
00135216	54	5D	33	9B	AF	80	BF	B1	3F	8B	D4	75	FE	CF	F8	87	T]3.....?..u....
00135232	AB	D8	B3	7D	04	DA	73	81	F9	D4	72	7E	D0	BF	F0	50	...}..s...r~...P
00135248	AD	28	1F	ED	7F	F8	27	BF	83	35	E5	5E	A7	42	F8	CD	.(....'.5.^..B..
00135264	60	DB	BE	82	EA	D2	2F	D4	D7	DA	EB	A8	83	DE	9C	75	`...../.....u
00135280	01	83	CF	6A	E6	9E	63	45	7C	58	78	3F	FC	0D	7E	53	...j..cE Xx?..~S
00135296	47	24	32	F9	3D	A6	FF	00	0F	D5	1F	08	4F	FB	67	FE	G\$2.=.....O.g.
00135312	D3	DA	51	03	C4	BF	F0	4C	BF	DA	77	00	65	DF	43	F1	..Q....L..w.e.C.
00135328	37	86	B5	35	5F	5C	7F	A7	46	CD	F9	56	44	DF	F0	51	7..5_...F...VD..Q

Each freelist trunk page has its own header, which breaks down as follows.

Freelist Trunk Page Header

OFFSET	SIZE	DESCRIPTION	TALK.SQLITE
0	4	Next Freelist Trunk Page	00 00 00 00 = This is the only one
4	4	# of Freelist Pointers on Page	00 00 00 03 = 3 Pointers (Next Three Groups of 4)
8	4	1 st Freelist Pointer	00 00 00 25 = 37
12	4	2 nd Freelist Pointer	00 00 00 24 = 36
16	4	3 rd Freelist Pointer	00 00 00 23 = 35



If the first four bytes equal zero, then the first freelist trunk page is the only one in the database and there will be no other freelist pages except those referenced within this header. If the value is non-zero, then converting the hex value to decimal would provide the page number of the next freelist trunk page. The formula above would take us to that next page and subsequently more freelist pages.

The next four byte set indicates how many freelist page references or pointers are to follow. In this case, there are 3 freelist leaf pages that are referenced within this trunk page. They will be the next three (4) byte values.

You can also open the talk.sqlite database with sqlparser.pyw. Then analyze and get Freelist info.

The next 3 four-byte blocks identifying the freelist leaf pages convert to decimal values 35, 36, and 37. Using the above formula, we can find the offset to these freelist pages.

Freelist pages referenced by the trunk page	Offset to page in database
37	147,456
36	143,360
35	139,264

The screenshot below is the page header for freelist page 35 found at offset 139,264:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0123456789ABCDEF
00139264	0D	05	A4	00	17	02	01	06	0F	D0	0F	9E	0F	6C	0F	311.1
00139280	0E	F5	06	39	05	FE	05	CB	05	2F	05	63	04	C0	04	F4	...9...../.c....
00139296	04	80	04	4C	04	09	03	D7	03	A0	03	69	03	17	02	E8	...L.....i....
00139312	02	BA	02	8B	02	01	00	00	00	00	00	00	00	00	00	00
00139328	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00139344	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00139360	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00139376	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

This freelist leafpage potentially holds deleted data from the database. The first record encountered from the start of page 35 will be found at offset 139,777 (139,264 + 513 Bytes offset from start of page). There are 23 records in total stored within this page. To view the first record encountered in the page, go to offset 139,777.

First Record from the start of Freelist Page 35 (First cell content area)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0123456789ABCDEF
00139776	00	81	07	17	12	00	01	01	01	01	01	05	01	00	07	01
00139792	25	0F	81	33	00	00	04	05	03	01	01	01	3D	EE	23	3A	%..3.....=.#:
00139808	EB	01	40	B0	97	CC	EA	85	83	C6	00	32	37	31	37	35	..@.....27175
00139824	31	34	30	32	33	30	39	53	2F	70	72	69	76	61	74	65	1402309S/private
00139840	2F	76	61	72	2F	6D	6F	62	69	6C	65	2F	41	70	70	6C	/var/mobile/Applications/A6C14BA
00139856	69	63	61	74	69	6F	6E	73	2F	41	36	43	31	34	42	41	ications/A6C14BA
00139872	41	2D	46	35	32	44	2D	34	39	33	35	2D	39	38	32	45	A-F52D-4935-982E
00139888	2D	33	44	43	39	38	31	39	31	31	41	32	41	2F	74	6D	-3DC981911A2A/tm
00139904	70	2F	5F	38	32	31	38	2E	6D	34	61	2D	16	11	00	01	p/_8218.m4a-....
00139920	01	01	01	01	05	01	00	01	02	25	0F	00	00	00	04	04%.....

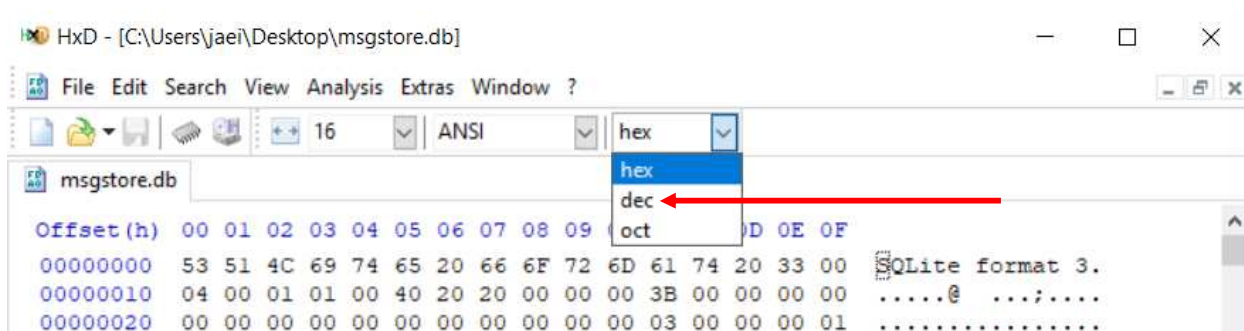
Preceding the record is a description header for the cell. We can use the terms cell and record interchangeably as they both refer to the same thing. This header tells the database what information and the length and type of values are to be found in each record row. Values can be different length integers, BLOBs, strings, floats, and NULLs.

Payloads of database records will vary in size from record to record. This allows the database to be as compact as possible. With the use of VarInts, the header can be dynamic and may not always have a specific length. Evaluating each byte in sequence will be essential in interpreting the header so that the payload will be parsed out correctly.

The first VarInt of the cell indicates the length of the entire record. In this case, the first byte is 0x81. Any value larger than 0x7F is the start of a variable integer, or VarInt.

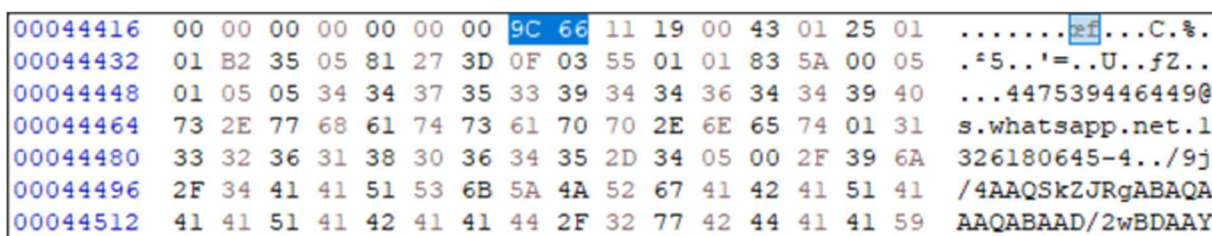
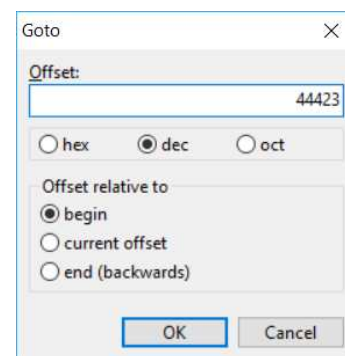
1.1.14 Varint

To see a VarInt in action, open msgstore.db found in the Day 1 sample files folder in HxD. Change the offset column to display decimal value offsets instead of hex values by selected “dec” from the dropdown menu in the ribbon bar.



Press CTRL-G to bring up the Goto dialogue box. Type in 44,423 and click OK to go to offset 44,423 in the file. This will take you to the start of a record.

The first two bytes from the offset is the VarInt. The key to decoding the VarInt correctly is to look at the first hex value in every byte. If the value is 8 or greater, then the following hex value is also part of the VarInt. If the value is 7 or lower, then that is the last byte in the VarInt.

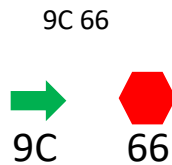


In this case, 9C66 is a two byte VarInt. The left nibble in 9C is greater than 7, indicating the following byte, 66, is also part of the VarInt. The left nibble in 66 is less than 8, so this is the last byte of the VarInt. These two bytes happen to indicate the length of the record. Record headers will be covered in greater detail later in this course. For now, we will convert this VarInt to an integer to determine the length of the cell content area.

You would be tempted to open Windows calculator in programmer mode, enter the hex value, and then tick decimal to make the conversion. But this would give you the wrong length. The length is not 40,038.

1.1.15 Converting a VarInt to a Hex Value

Remember, the key to decoding a VarInt is to look at the left nibble of each byte to determine how many bytes make up the VarInt.



Step 1: Convert all the hex values to binary

10011100	01100110
----------	----------

Step 2: Drop all of the indicator bits (the first most significant bit of each byte):

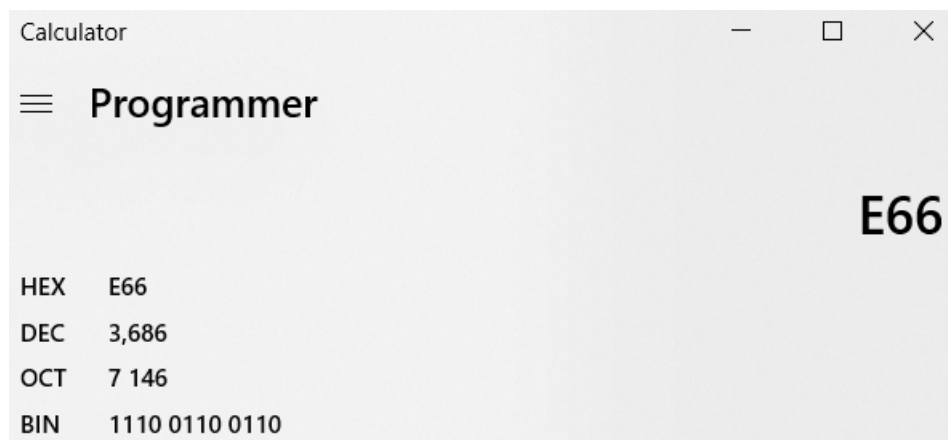
FROM	10011100	01100110	TO	00111100	1100110
-------------	----------	----------	-----------	----------	---------

Step 3: Starting from the right, take least significant bits from each neighboring byte to make a complete 8-bit byte (pad with zeros if necessary at the beginning)

FROM	0011100	1100110	TO	00001110	01100110
-------------	---------	---------	-----------	----------	----------

↓
0E 66

Converting 0E 66 to decimal gives us the value 3,686. As shown below in the Windows Calculator in Programmer Mode. This is the correct length of the cell content.



1.1.16 Back to our Freelist Page Example

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0123456789ABCDEF
00139776	00	81	07	17	12	00	01	01	01	01	01	05	01	00	07	01
00139792	25	0F	81	33	00	00	04	05	03	01	01	01	3D	EE	23	3A	%..3.....=.#:
00139808	EB	01	40	B0	97	CC	EA	85	83	C6	00	32	37	31	37	35	..@.....27175
00139824	31	34	30	32	33	30	39	53	2F	70	72	69	76	61	74	65	1402309S/private
00139840	2F	76	61	72	2F	6D	6F	62	69	6C	65	2F	41	70	70	6C	/var/mobile/Applic
00139856	69	63	61	74	69	6F	6E	73	2F	41	36	43	31	34	42	41	ications/A6C14BA
00139872	41	2D	46	35	32	44	2D	34	39	33	35	2D	39	38	32	45	A-F52D-4935-982E
00139888	2D	33	44	43	39	38	31	39	31	31	41	32	41	2F	74	6D	-3DC981911A2A/tm
00139904	70	2F	5F	38	32	31	38	2E	6D	34	61	2D	16	11	00	01	p/_8218.m4a-....
00139920	01	01	01	01	05	01	00	01	02	25	0F	00	00	00	04	04%.....

Since the first byte is larger than 0x7F, the second hex value will also be part of the VarInt. The second value is 0x07, not greater than 0x7F, meaning it is the end of the VarInt.

The value needing conversion is 0x8107. This will provide us with the length of the entire record.

Following the record length indicator is the ROW ID identifier. The ROW ID in this example is 0x17, or decimal 23. The cell/record can be found immediately following the ROW ID identifier.

rowid	_id	key_remote_id	key_from_me	key_id	status	needs_push	data
4	4	447539446449@swhatsapp.net	1	1324423002-3	5	0	0 Ho, here he is!
5	5	447539446449@swhatsapp.net	0	1325079055-1	0	0	0 See you at Satriale's later
6	6	447539446449@swhatsapp.net	1	1324423002-4	5	0	0 /s/4AAQSkZIRgABAQAAQAABAAD/2w8DAAYEBQYFBAYGBQYHBwYlChACgKjChC
7	7	447539446449@swhatsapp.net	1	1324423002-5	5	0	0 /s/4AAQSkZIRgABAQAAQAABAAD/2w8DAAYEBQYFBAYGBQYHBwYlChACgKjChC
8	8	447539446449@swhatsapp.net	1	1324423002-6	5	0	0 Got the tools for the job next week
9	9	447539446449@swhatsapp.net	0	1326141236-1	0	0	0 /s/4AAQSkZIRgABAQAAQAABAAD/2w8DAAYEBQYFBAYGBQYHBwYlChACgKjChC
10	10	447736298142@swhatsapp.net	0	1326134106-22	0	0	0 /s/4AAQSkZIRgABAQAAQAABAAD/2w8DAAYEBQYFBAYGBQYHBwYlChACgKjChC
11	11	447736298142@swhatsapp.net	0	1326134106-23	0	0	0 This is the vehicle. Silvio will be driving on the day
12	12	447539446449@swhatsapp.net	0	1326141236-2	0	0	0 /s/4AAQSkZIRgABAQAAQAABAAD/2w8DAAYEBQYFBAYGBQYHBwYlChACgKjChC
13	13	447539446449@swhatsapp.net	0	1326141236-3	0	0	0 /s/4AAQSkZIRgABAQAAQAABAAD/2w8DAAYEBQYFBAYGBQYHBwYlChACgKjChC
14	14	447539446449@swhatsapp.net	1	1326180645-1	5	0	0 /s/4AAQSkZIRgABAQAAQAABAAD/2w8DAAYEBQYFBAYGBQYHBwYlChACgKjChC
15	15	447539446449@swhatsapp.net	1	1326180645-2	5	0	0 You need a quiet word with Sil
16	16	447539446449@swhatsapp.net	1	1326180645-3	5	0	0 Let me know when the work is done
17	17	447539446449@swhatsapp.net	1	1326180645-4	5	0	0 /s/4AAQSkZIRgABAQAAQAABAAD/2w8DAAYEBQYFBAYGBQYHBwYlChACgKjChC

VARINT Analysis

Use the VarInt calculator to convert VarInts to integers and string/Blob lengths. VarInts will automatically be converted to a string or Blob length depending on the Varint value.

1. What is the Variable Integer length of 0x8107? _____

The record has its own header, and each value within the header identifies lengths of data and types. The first byte of the cell content area provides the length of the record header. 0x12 indicates the header is 18 bytes in length.

1.1.17 Record Headers

Each hex value has a unique type associated with it depending on its value:

Serial Type <i>All serial types are decimal values</i>	Content Size <i>How many Bytes you read</i>	Meaning
0	0	NULL
1	1	8-bit
2	2	Big-Endian 16-bit
3	3	Big-Endian 24-bit
4	4	Big-Endian 32-bit
5	6	Big-Endian 48-bit
6	8	Big-Endian 64-bit
7	8	Big-Endian 64-bit Floating Point
8	0	Integer Constant 0. Only Available for Schema Format 4 or Higher Only shows in Record Header, not in record payload.
9	0	Integer Constant 1. Only Available for Schema Format 4 or Higher Only shows in Record Header, not in record payload.
10,11		Not Used
N ≥ 12 Even	(N-12) / 2	BLOB that is (N-12)/2 Bytes in Length
N ≥ 13 Odd	(N-13) / 2	String using the database encoding scheme (N-13)/2 bytes in length

Example Company.db from Day one.

2 000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 23 01 08 00 15 #00 0

2 016 17 04 01 09 27 4A 6F 68 6E 53 6D 69 74 68 4D 7F 000 'JohnSmithM

2 032 1C 80 23 53 61 6C 65 73 20 4D 61 6E 61 67 65 72 #Sales Manager

Record Length	ROW ID	Header Length		First	Last	DOH	Age	Gender	Title
VARINT 23 Dec 35 Bytes Does Not Include VARINT or ROW ID	01 First Record In this case	VARINT 08 Convert to Decimal 8 Bytes Offset 08-27	00 NULL Place Holder	VARINT 15 = 4 Length String "John"	VARINT 17 = 5 Length String "Smith"	04 Big Endian 32 Bit	01 1 Byte	09 Integer Constant 1	VARINT 27 = 13 Length String

Last Name: Smith
Varint 17 = Dec 23
(23 - 13) / 2 = 5

VarInt C... — □ ×

File Edit Help

13

Clear VarInt Integer Length

1.1.18 Overflow Page

If data needs more room then the SQLite Database can make use of overflow pages. These can contain binary large objects (BLOBs) i.e. Apple Property Lists, XML Files, and Media files like a jpeg picture file.

The last 4 bytes of a page may reference an overflow page. Below the end of the page shows a four byte value of 0x00 00 00 22 that is a page number (34), directing you to the overflow page.

```
Offset(d) 00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15
00016336 5D C4 D7 86 F8 FE EF F6 D9 4D 66 19 BF 67 9B 8F ]Ä*+øþiöÛMf.¿g>.
00016352 D9 5E E3 C0 C2 0C 4D 6B E3 58 F5 74 BF 37 7B DB Û^ääÂ.MkãXöt¿7{Û
00016368 79 59 6C C9 8F CB 2B B3 00 A6 E0 43 00 00 00 22 yYlÉ.Ë+°.!àC..."
00016384 0D 00 00 00 0A 0B 9D 00 0F C9 0E 03 0D D1 0D 64 .....É...Ñ.d
```

1.1.19 Binary Large Objects and Base 64 Encoding

Base64 encoding was designed for storing or sending any kind of binary (hex) data in a format that will not contain any illegal characters that would cause a computer to interpret the data as instructions. For example, the first 32 characters in the ASCII table are instructions for computers, like cancel and escape. If those characters would appear in the data stream, the computer will be confused and start to misbehave. E-mail is another application that makes use of Base64 encoding. All data in Base64 is converted to only use the hex values of the characters A-Z, a-z, 0-9 and the + and / symbols. (Some URLs and filenames replace + and / symbol with _ and -). Base64 encoded data is easily detected because the stored data will most likely end with one or two = signs at the end of the string. In Base64, the 8-bit encoded data is converted to 6-bit encoded data.

Base 64 Table

Value	Char	Value	Char	Value	Char	Value	Char	Value	Char	Value	Char
0	A	11	L	22	W	33	h	44	s	54	2
1	B	12	M	23	X	34	i	45	t	55	3
2	C	13	N	24	Y	35	j	46	u	56	4
3	D	14	O	25	Z	36	k	47	v	57	5
4	E	15	P	26	a	37	l	48	w	58	6
5	F	16	Q	27	b	38	m	49	x	59	7
6	G	17	R	28	c	39	n	50	y	60	8
7	H	18	S	29	d	40	o	51	z	61	9
8	I	19	T	30	e	41	p	52	0	62	+ -
9	J	20	U	31	f	42	q	53	1	63	/ _
10	K	21	V	32	g	43	r				

The conversion to base64 is done by taking 3 bytes of data at a time, consisting of 24 bits, and converting those 24 bits to four 6-bit values. (4 x 6 = 24 so the number of bits is constant). This allows for the storage of any hex value in a format acceptable by all computers and software.

Say we want to send a sequence of hex values as an attachment. In this example, we will use a hex string of 1B054C. Hex values 1B and 05 are instructions and those could cause problems with the computer that is either sending or receiving the content. To combat that, the sequence will be converted to base64 and sent.

The conversion process is demonstrated in the following table:

Text	ÿ	ø	ÿ
ASCII Hex values	FF	D8	FF
Bit pattern	1 1 1 1 1 1 1 1	1 1 0 1 1 0 0 0	1 1 1 1 1 1 1 1
6bit Dec values	63	61	35
Base64-encoded			

6 Bit Column Headers Binary to Base 64

32	16	8	4	2	1
1	1	1	1	1	1

Total: 63

32	16	8	4	2	1
1	1	1	1	0	1

Total 61

32	16	8	4	2	1
1	0	0	0	1	1

Total 35

32	16	8	4	2	1
1	1	1	1	1	1

Total: 63

If the data that is decoded is not in an even 3-byte length, the last 3-byte array is padded with = signs. So, if the last group of 3 bytes only has 1 byte of information, two = signs will be added to the group. If the last group of data has 2 bytes of information, one = sign is padded on to the end.

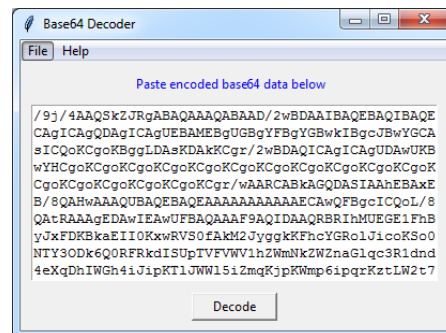
Base64 Data Analysis

Open **msgstore.db** in SQLite Expert (or other database browser). This database is or was used by WhatsApp on Android to store chat messages.

View the messages table and the data column for the records within. What can you determine is stored in the data field for some of those records? How did you come to this conclusion?

Click a record where the data field contains `/9j/`. Click the ellipses (...) button. Use CTRL-A to select all and then copy the contents to the clipboard.

Open **Base64 Decoder.pyw** and paste the contents of the clipboard into the large text box and click the Decode button. Save the decoded file to the Desktop with a `.jpg` file extension.



Are you able to open and view the file you saved to the Desktop? _____

1.1.20 Additional Sources of Data

There are other files for you to examine that are associated with the SQLite Database. If you remember from the SQLite Database File header there are two offsets (18 and 19) that identify the type of associated files.

OFFSET	SIZE	DESCRIPTION	ADDITIONAL INFORMATION
0	16	Header string: "SQLite format 3"	
16	2	Page Size – Big Endian	
18	1	File Format Write Version 1 = Legacy / 2 = WAL	01 = Legacy Rollback Journal 02 = Shared Memory (SHM)/Write Ahead Log (WAL)
19	1	File Format Read Version 1 = Legacy / 2 = WAL	01 = Legacy Rollback Journal 02 = Shared Memory (SHM)/Write Ahead Log (WAL)

The Shared Memory File and the Write Ahead Log File or the Rollback Journal. If you copy a SQLite database from the extraction. Make sure you also copy these files out along with it.

The image, shown right, should tell you something about the WAL File and its importance.

sms.db	4 KB
sms.db-shm	32 KB
sms.db-wal	2,197 KB

1.1.21 Python: What SQLite Related Data do we need to know?

- Database name
- Path to the database
- Database Table Names
- Relational Aspect between Tables
 - Contact/User ID
- Column Headers
 - Date and Time Stamps
 - Location Data
 - Flagged items:
 - Calls: Dialed/Outgoing, Received/Incoming, Missed, Voicemail
 - Messages: Sent, Received

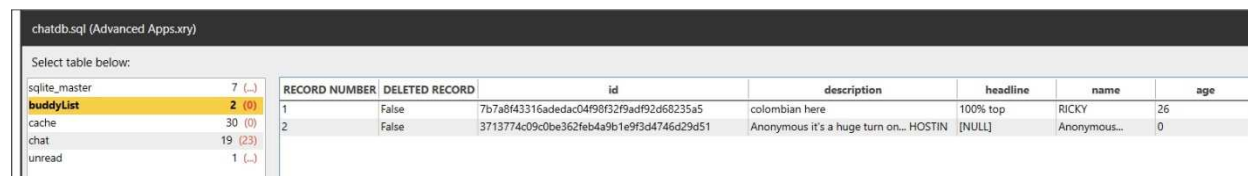
1.1.22 Chatdb Analysis

Chat messages are located in the **chat** table within the database. The name of the correspondent is not used within this table, only their **buddy** ID is used.

Advanced Apps - XAMN

<

But there is a **buddyList** table located with the database. The **buddyList** contains the **id** along with a **name**. The "id": "name" would be perfectly suited for a Python dictionary.



The screenshot shows the XAMN database viewer interface. The 'chatdb.sql' file is open, and the 'buddyList' table is selected. The table contains 2 records. The columns are: RECORD NUMBER, DELETED RECORD, id, description, headline, name, and age.

RECORD NUMBER	DELETED RECORD	id	description	headline	name	age
1	False	7b7a8f43316adedac04f98f32f9ad92d68235a5	colombian here	100% top	RICKY	26
2	False	3713774c09c0be362feb4a9b1e9f3d4746d29d51	Anonymous it's a huge turn on... HOSTIN	[NULL]	Anonymous...	0

1.1.23 Python and SQLite

Now it's time to dive into parsing data from SQLite databases. This is similar to the previous section dealing with PLists as we need to locate the database, export it, load it, and then iterate through it to parse the data we are looking for. The SQLite3 module allows for the connection to a database and the reading of records from various tables.

1. Import the modules we need for SQLite3 and converting date time stamps. OS can be used for creating directories to store the output csv file. Datetime can be used for epoch time stamp conversion
2. Create Global Variables to identify the author, version, and description of the script.'
3. Open the csv file that we want to write to.
4. What data do we want to write to the csv file.
5. Create a Dictionary named buddies.
6. Connect to the SQLite Database identified
7. Create variable for the connection cursor to go through the database.
8. Execute the cursor to go through the identified records in the identified table.
9. Create a for loop to go through each row and obtain the data.
10. Identify buddies as the name item 1 as opposed to the id item 0
11. Connect to the chat table and select the following columns: msg, stamp, from, and buddy.
12. Create a for loop to obtain the data from these columns and identify them as listed below:
 - a. msg
 - b. stamp (and convert the date time stamp to local time)
 - c. identify the direction (from column) that the message was sent. Using an if/else statement to identify the flag as either to or from.
 - d. Identify the name and not the id of the person
13. Print the columns msg, stamp, direction, and name to Idle.
14. Print the columns msg, stamp, direction, and name to the csv file opened at the beginning.
15. Close the file
16. Close the connection to the database.

If you copy a script to apply it to another database then remember you will need to identify the tables and record column headers.

1.1.24 Python: Chatdb.sql to CSV

```
# Copy the Chatdb.sql file to your C:\Temp folder

import os

import sqlite3

import datetime

__contact__ = "MSAB"

__version__ = "1"

__description__ = "chat db to csv"

f = open(r"C:\Temp\export.csv", "wt")

f.write("MESSAGE,TIMESTAMP,DIRECTION,NAME\n")

buddys = {}

conn = sqlite3.connect(r"C:\Temp\chatdb.sql")

c = conn.cursor()

c.execute("Select [id], [name] from buddyList")

for row in c:

    buddies[row[0]] = row[1]

c.execute("Select [msg], [stamp], [from], [buddy] from chat")

for row in c:

    msg = row[0]

    stamp = datetime.datetime.utcfromtimestamp(row[1])

    if row[2] == 1:

        direction = "TO"

    else:

        direction = "FROM"

    name = buddies[row[3]]

    print(msg, stamp, direction, name)

    f.write("{}{},{},{}\n".format(msg, stamp, direction, name))

f.close()

conn.close()
```

Python Resources

Please be aware that in class we will be using version 3.x and many of these may use an older version.

MSAB Customer Portal – Forensic Forum- Python folder

MSAB's Advanced Application Analysis Course www.msab.com/training/advanced-apps-analysis/

Pycharm

<https://www.jetbrains.com/pycharm/>

Notepad++

<https://notepad-plus-plus.org/>

Visual Studio Code

<https://code.visualstudio.com>

Books:

There are plenty of books on Python. Don't forget about children's books that focus on creating games. There are also some Python Forensic books. For Forensic books be sure to preview the table of contents (with Amazon) to see if it has topics relative to the work that you do. Some may focus on network or computer forensics.

Courses:

There are some online courses through Coursera, Udemy, Youtube, and more.

Here are a couple to consider:

coursera.org

<https://www.coursera.org/learn/python>

Google

<https://developers.google.com/edu/python/>

Youtube

<https://www.youtube.com/watch?v=N4mEzFDjqtA>

<https://www.youtube.com/user/thenewboston>

https://www.youtube.com/watch?v=HBxCHonP6Ro&list=PL6gx4Cwl9DGAcbMi1sH6oAMk4JHw91mC_

Websites:

www.python.org

<https://www.pycon.org/>

<http://pythonineducation.org/en/>

<http://www.sphinx-doc.org/en/stable/>

https://www.tutorialspoint.com/python/python_gui_programming.htm

<http://python-forensics.org/>

https://www.tutorialspoint.com/python_forensics/index.htm

<http://opensourceforu.com/2016/11/python-programming-digital-forensics-security-analysis/>