Rebuilding APP Databases to get back deleted data

Analysing freepages in SQLite databases

Martin Westman – Micro Systemation

IMF Münster 2014

MICRO SYSTEMATION

Warning! Achtung! Attention! اهتماماهتمام

- This is a pretty Technical Session:
  - We will talk Hex, pages, offsets and headers
  - If you are new to SQLite or databases, this will be a compressed complicated session
  - Main mission will be to show you the possibilities that we have, not to be experts
  - The time will not allow us for more deeper knowledge in the SQLite format and functions
  - You are free to follow my presentation on you own computer, files are available on USB drives

MICRO SYSTEMATION

Objectives

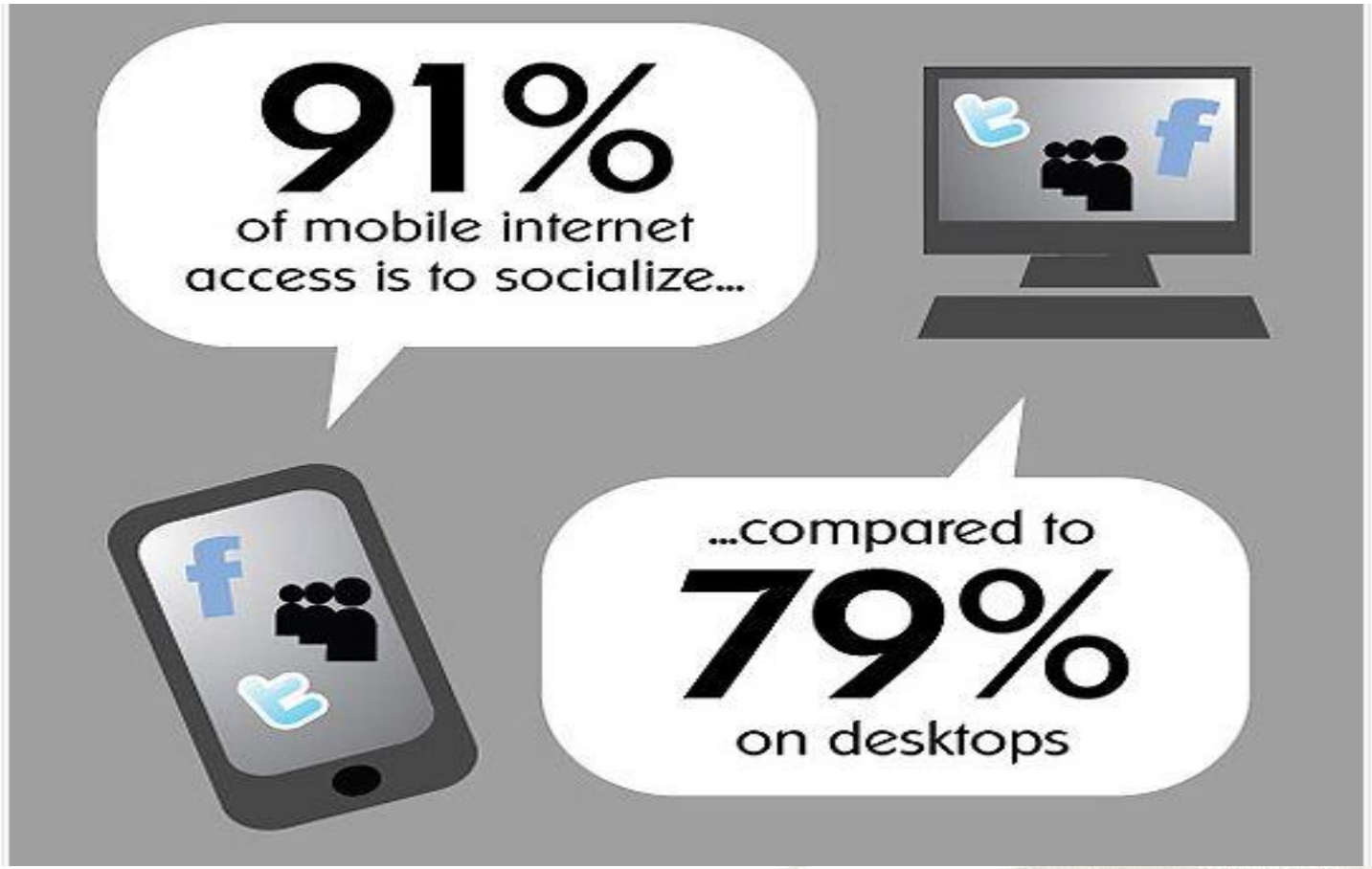
- By the end of this session we (hopefully) have:
  - Basic knowledge of SQLite3 databases
  - Worked with SQLite database viewer
  - Work with Hex reader and editor
  - Created a database with previously deleted material from abandoned Freepages in a database

MICRO SYSTEMATION

User app market

“There’s an app for that”

MICRO SYSTEMATION



Over 200 million (1/3 of all users) access Facebook from a mobile device and 91% of all mobile Internet use is “social” related

MICRO SYSTEMATION

Example of current iPhone ‘App’ support in forensic softwares

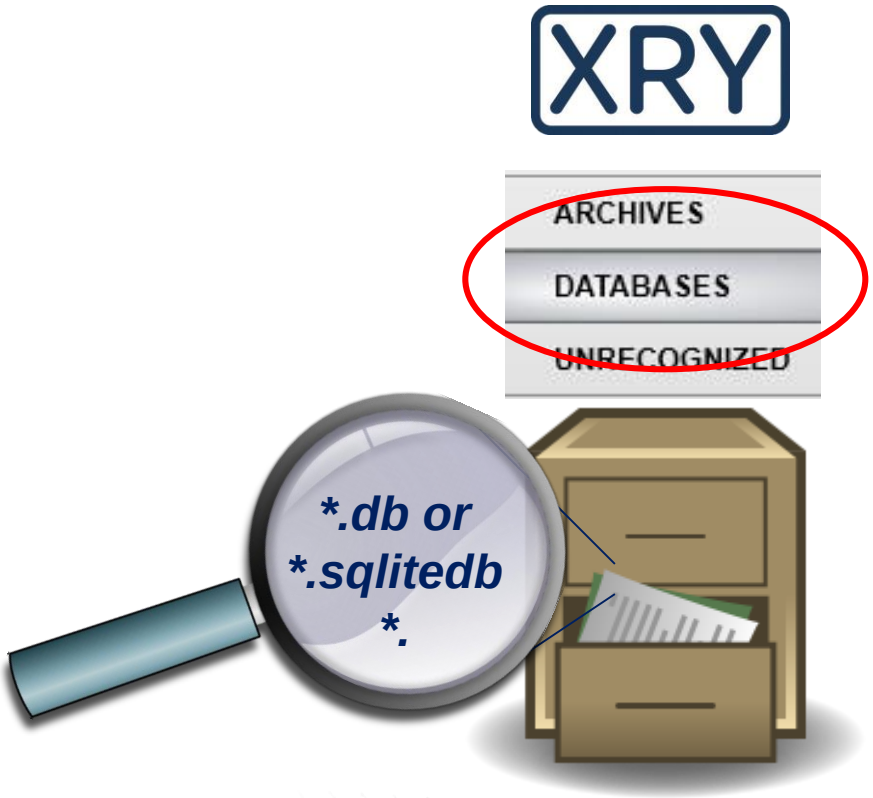
- AIM (AOL Instant Messenger)
- eBuddy
- Facebook
- Find My iPhone
- Flickr
- Fring
- Google Earth
- Google Maps
- Google Mobile
- Google Translate
- hitta.se
- ICQ Free
- Kik Messenger
- MCleaner (Cydia)
- MySpace
- NAVIGON Mobile Navigator
- Nike+ GPS
- Nimbuzz
- PingChat!
- RunKeeper
- Search Bing
- Skype
- TomTom
- Twitter
- Twitterrific for Twitter
- Viber
- WhatsApp
- Yahoo! Messenger
- Yahoo! SearchYouTube
- Weechat
- Kakao talk

Databases

- Databases are used to store larger amounts of data in an orderly and efficient fashion
- You will find databases in everything from huge datacenters like the TAX Revenue organizations and call centers and 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

MICRO SYSTEMATION

Smartphone SQLite Databases



- Compact
- Cross platform
- Reliable
- Zero configuration (all in one file..-ish)

Embedded SQL database engine

MICRO SYSTEMATION

Investigating Unsupported Apps

- Last year we recovered encrypted data from app's database that are unsupported by forensic tools.
- But what if we want to search for deleted data in unsupported apps?
- Deleted data will not be reported by database viewers since the data is just "deleted" and no longer known to the database.
- Let's have a BASIC introduction to SQLite and how it stores data to see if we can get deleted data out from the database structure

SQLite database structures

- Since SQLite is Zero-configuration all information is found in one single file.
- Data is structured in a B-Tree format
- B-Tree formatting of data can be compared to a tree (like a maple tree for example) structure
- You can think of the branches (Interior Pages) that points on the actual data (leaf Pages with the actual information, or payload written on them)
- The leaf Pages have also pointers to where on the actual leaf we will find the information we are looking for,(which tip that holds our payload)

The records

- A record is a chunk of information in the database, such as a row of data in the database
- A record is made up of a header and payload data
- All values in a header are varint’s
- The first value is the header size, all other values describes the data and the in the following payload
- The payload is then like one long line of data that is cut up accordingly to the header and put in the right place in the table inside the database

The Pages holding the data

- The SQLite file is comprised of “Pages” of a specified size containing pointers and payload data
- The first Page additionally contains a database header of 100 bytes
- Pages can be either of the “interior” format (the branches in our tree comparison) holding pointers to data, or the “leaf” format holding actual payload data (the leaves)
- A Page holding inactive or “deleted” data it is called a freepage

Math within the file and Pages

- All addressing in the file and Pages are always in Hex
- References to Offset in Pages are always counted from the start of the individual Page, not offset in the file itself
- Start of individual Pages can be found by multiplying the Page number with the Page size , but remember to deduct 1 from the Page number to get to the start of the Page (for example to find start of page 34 count (34-1)*Page size)

In the header of the File

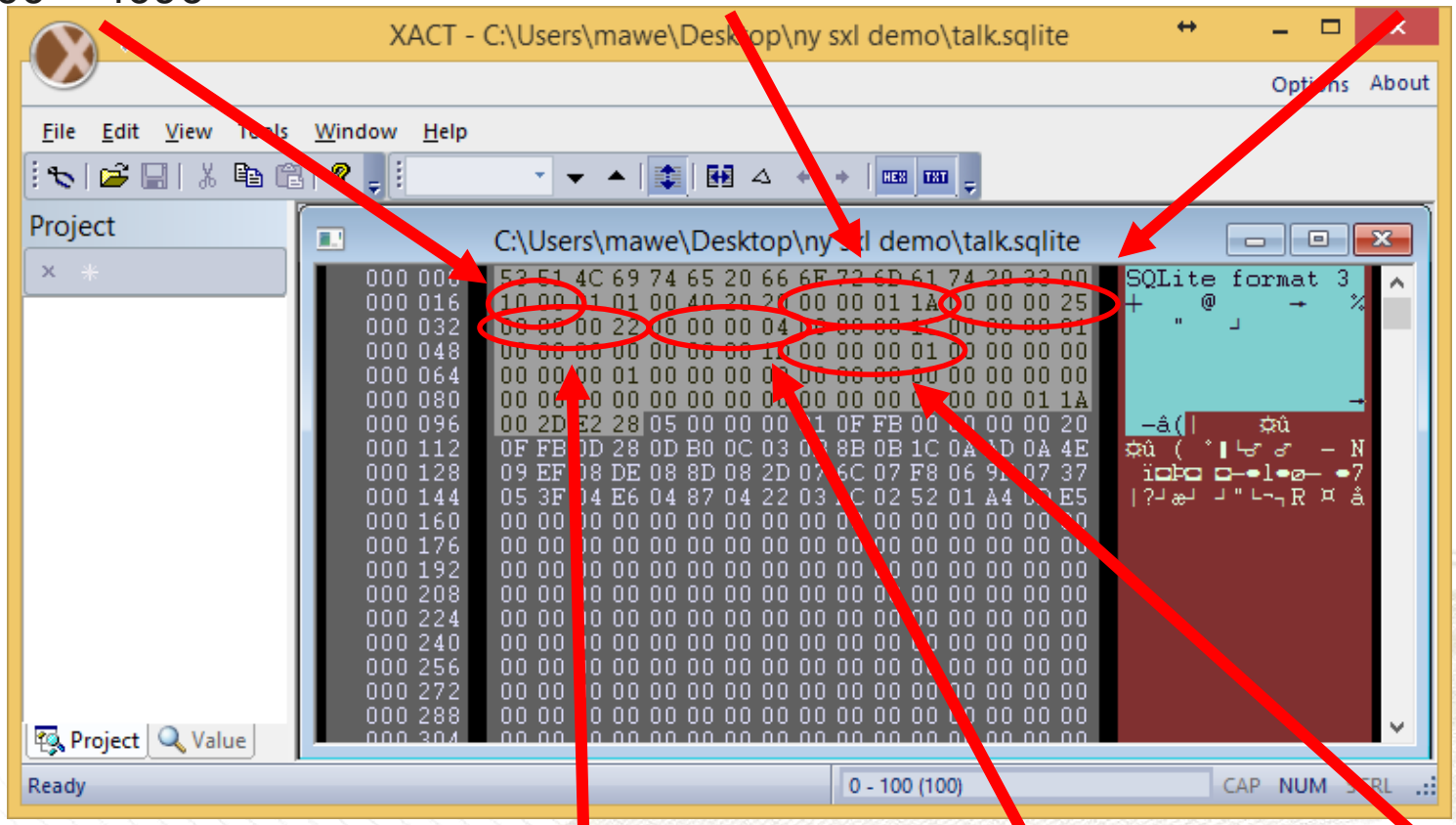
- In the 100 Byte file header we will find a lot of information about the data in the file, like the size of, and how many Pages it consists of and other useful information
- We will also find where to find the “freelists trunk page” in the header, as well how many freelist Pages we can expect to find
- The freelist trunk Page holds information of old abandoned data pages, and pointers to where to find the freelist Pages containing the actual deleted data

Inside the file header

@16 2 B indicating the Page size 0x1000 = 4096

@24 4 B counter how many times data has changed in the file 0x11A = 282

@28 4 B How many Pages the database consist of 0x25 = 37



@32 4 B Where to find the freelist trunk Page 0x22 = 34

@36 4 B How many freelists the database contains 4

@56 4 B The encoding scheme used 1 means UTF8

MICRO SYSTEMATION

The header of Pages

B-tree Page Header Format

Offset	Size	Description
0	1	A flag indicating the b-tree page type A value of 2 means the page is an interior index b-tree page. A value of 5 means the page is an interior table b-tree page. A value of 10 means the page is a leaf index b-tree page. A value of 13 means the page is a leaf table b-tree page. Any other value for the b-tree page type is an error.
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 the cell content area. A zero value is used to represent an offset of 65536, which occurs on an empty root page when using a 65536-byte page size.
7	1	Number of fragmented free bytes within the cell content area
8	4	The right-most pointer (interior b-tree pages only)

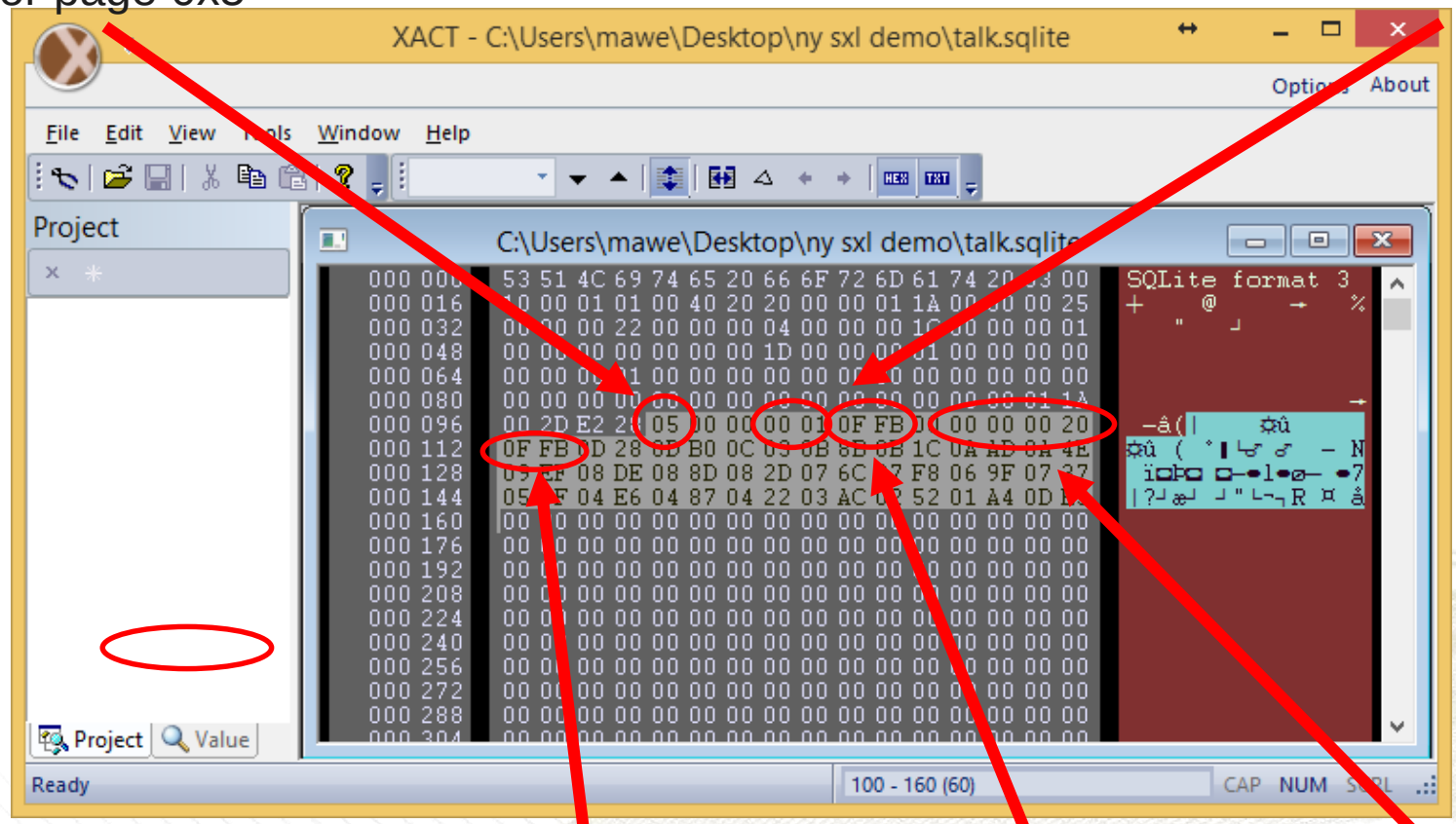
The header of Pages

- The first Byte in the Page header indicates what type it is: Leaf or Interior
- The first 8 Bytes (leaf Pages) or 12 Bytes (interior Pages) are followed by pointers to the Cells
- A Cell can contain a pointer to data or hold the actual payload for the database
- The cell pointer array (that points to the actual content of the cell) is found just after the 8 or 12 byte Page header
- Cell data is stored in the end of the Page to leave space for more pointers in the beginning of the file if needed.

Inside the page header

@0 1 B indicating the Page is a interior page 0x5

@5 2 B Number of Cells in this Page 0x1



First of the cellpointer arrays (only 1 here) 0xFFB = 4091

@7 2 B Offset to cell content area

@12 4 B (only interior Page) the rightmost Page pointer 0x20 = 32

MICRO SYSTEMATION

The freelist page(s)

- If you delete a lot of data, that spans over a full page or if there is a new version of the page it will be pushed out of the database and referred to as freepage
- Freepages are complete database pages just “inactive” and can (and will be) reused for storage of new data
- Normally one of the old used data pages will become a freelist trunk page

The freelist trunk page

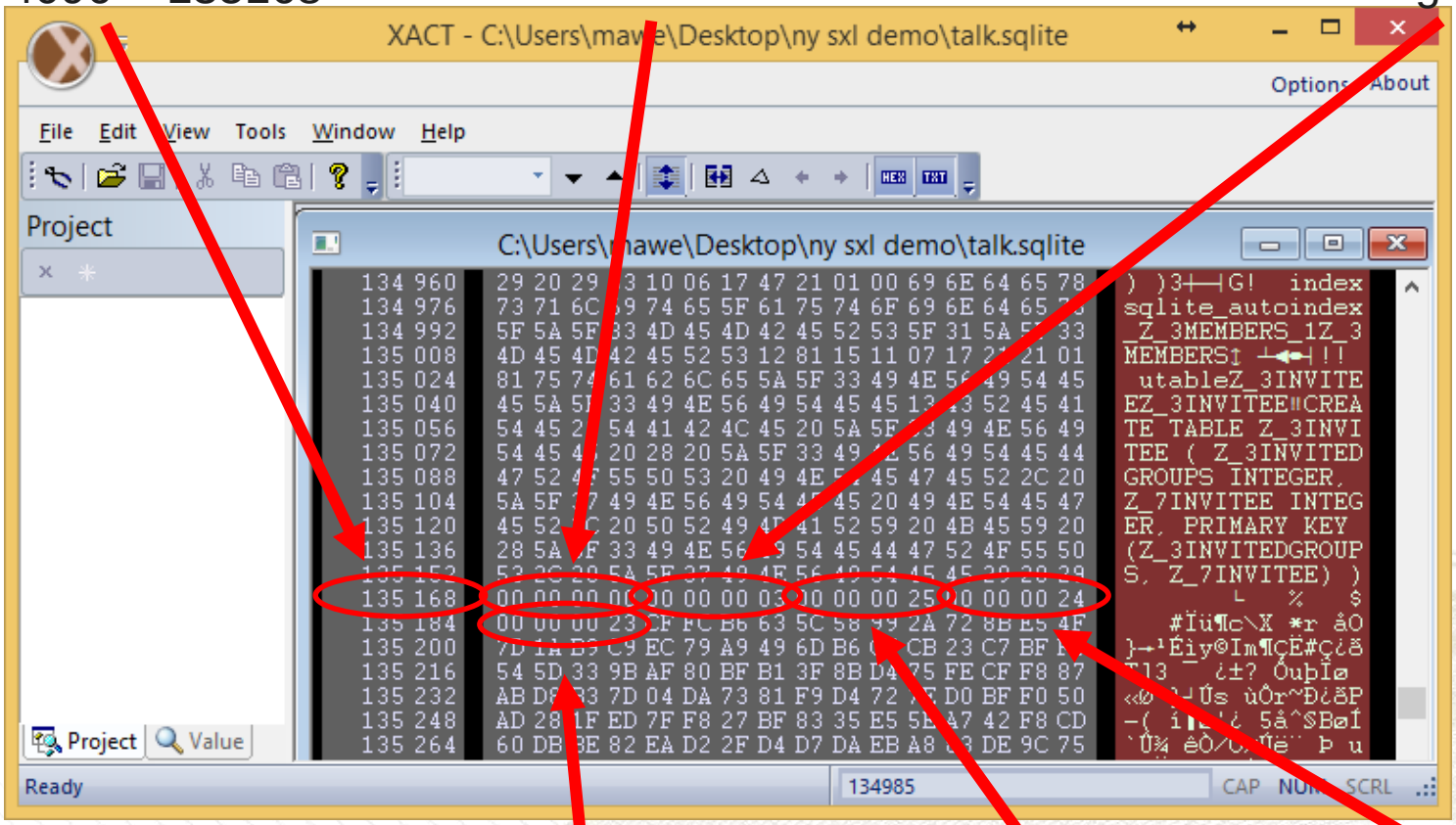
- The freelist trunkpage(s) describes where to find the freelists, the Pages that holds the “inactive” or deleted data
- The first 4 Bytes contains reference to the next freelist trunk Page, if all 0’s there is only one list
- The next 4 Bytes describe how many references to freelist pages there are stored on this trunk page
- After that the actual Page numbers for the freelists follows in 4 Byte sequences

The freelist trunk page header

The address found in the header of the file (34-1)*4096 = 135168

@0 4 B reference to next freelist trunk Page 0x00 00 00 00 (no more list)

@4 4 B number of freelist pointers on this Page 0x3 = 3



3rd freelist Page pointer 0x23 = 35

1st freelist Page pointer 0x25 = 37

2nd freelist Page pointer 0x24 = 36

MICRO SYSTEMATION

Lets get to work

- The aim for this session is to locate the freelists in a database, and create an own database with the content of the freelists, and trick SQLite that it is a live database
- So the first thing we need to do, is to open our database of choice: Line Talk, and look for how many freepages it consist of and where they are found
- So please drag the talk.sql file on the desktop to the SQLite expert personal shortcut

Locating the missing data

- If we look now in the PK (Primary Key) we can se that the values below 29 is missing, as well as 45
- That means these rows are missing, or deleted data
- SQLite populates the column declared as Integer Primary Key (in this case the PK column) with the row id when the row is created (in most cases)
- Close down SQLite expert and drag the talk.sql to the red XACT icon on the desktop

Now, lets try to find the missing data

- Start by checking the 4 Byte value at offset 32
  - The first freelist trunk Page 0x(00 00 00 22) = Page 34
- Then check the number of freelist at offset 36
  - Total number of freelists are 0x(00 00 00 04) = 4 Pages
- Now we need to check the size of the Pages at offset 16 0x(1000) = 4096
- Now we can calculate where to find the first freelist trunk page with the previous mentioned algorithm
  - (First freelist trunk Page – 1) * Page size
  - (34-1) * 4096 = 135168

Analysing the freelist trunk Page

- Start by checking the 4 Byte value at offset 0
  - The first 4 Byte will point out any following freelist trunk Pages, if all are 00 this is the last / only trunk Page (00 00 00 00)
- Now we need to find total of number of freelists at offset 4, 4 Byte
  - Total number of leaf freelist (containing data) is 00 00 00 03
- Now follows in increments of 4 Byte the Page numbers of the freelists
  - 00 00 00 25, 00 00 00 24, 00 00 00 23 in Hex, remember?
  - That means Pages 37, 36 and 35 are freelist Pages
  - So the freelist are found at: 147456, 143360 and 139264

Analysing the freelist Page

- Now we need to find a record (or payload cell, or by other word, a missing row in our database)
  - Where to look? Remember that the Cells (payload) of data was stored from the end of the Page
- So lets go to the end of the first freelist Page and work our self upwards until we find our first record
  - Press CTRL + G and type in the end of Page 35 / the beginning of Page 36 (143360)
- Lets scroll upwards till we find the first record
  - That means when it is just filled with 00 above we have found our first record (139777)

Well, that's not readable at all… or is it?

- For every single Cell (row of data) there is a description header, or Record Format that tells the database what information and the length and type of the value in the actual database row
- It can be different length Integer's, BLOB's, Strings, Floats and NULL values
- And if that was not complicated enough, all values within cells are stored as Varints…
- …what the heck is that???

The beautiful and exciting world of Varints

- A varint is a VARiable length INTeger (of course… ☺)
- A varint can be 1-8 Byte long integer stored in 1-9 Bytes
- Hence it can be of different length, depending of how long, or big, the integer is
- The interesting thing with varints is the way they store data. To see how, we need to break it down into binary
- The maximum value of a varint byte is 127, the reason for that is because the first (or most significant bit) is used to express if there is a following byte of information in the varint.

Varint structure

- Lets say we are storing the value of 135 using varint
- Converting 135 to Hex will give us 0x87
- Converting Hex 87 to binary for storage would normally give us 1000 0111
- But since in varint we can only use the 7 last bits for actual value we have to store it as:
- 1000 0001 0000 0111 hence using 2 Bytes to store it
- The red digit of 1 means the following byte needs for calculating our stored value, the 0 means it is the last byte in this varint
- Varint tip: if the first Hex char is >8, the following Byte is part of the varint value

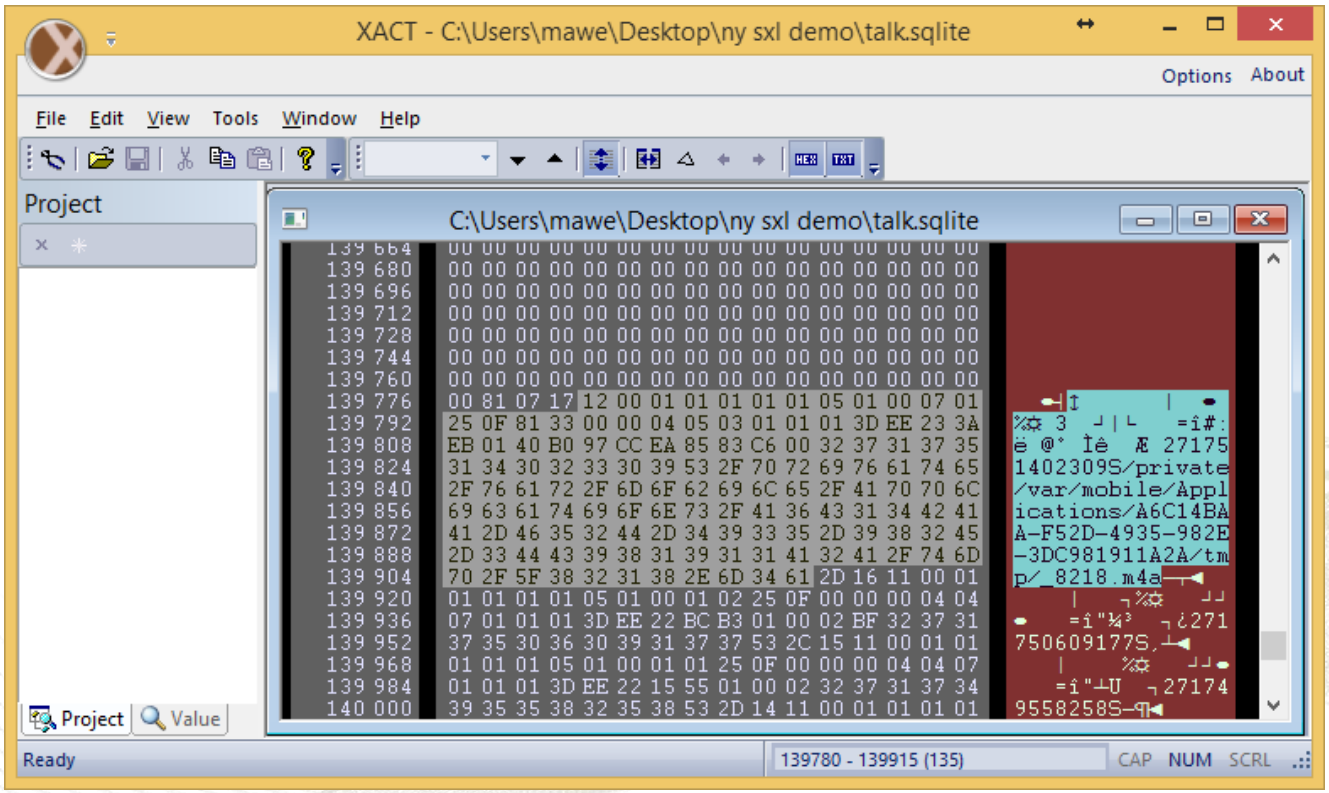
MICRO SYSTEMATION

Converting Varint values

- Lets say we want to store the value 718 using varint.
- 718 Dec converted to Hex is 02CE
- Binary of Hex 02CE is 0000 0010 1100 1110
- To store that in a varint it will be stored as
- 1000 0101 0100 1110 or the Hex 854E
- A longer value like 83254341 would be in binary
- 0000 0100 1111 0110 0101 1100 0100 0101= 04F65C45
- Varint value of that would be :
- 1010 0111 1101 1001 1011 1000 0100 0101= A7D9B845

Now we have all knowledge we need!

- Lets go back to database file in XACT and try to interpret the values.



MICRO SYSTEMATION

Interpret the varint values

Serial Type Codes Of The Record Format

Serial Type	Content Size	Meaning
0	0	NULL
1	1	8-bit twos-complement integer
2	2	Big-endian 16-bit twos-complement integer
3	3	Big-endian 24-bit twos-complement integer
4	4	Big-endian 32-bit twos-complement integer
5	6	Big-endian 48-bit twos-complement integer
6	8	Big-endian 64-bit twos-complement integer
7	8	Big-endian IEEE 754-2008 64-bit floating point number
8	0	Integer constant 0. Only available for schema format 4 and higher.
9	0	Integer constant 1. Only available for schema format 4 and higher.
10,11		*Not used. Reserved for expansion.*
N≥12 and even	(N-12)/2	A BLOB that is (N-12)/2 bytes in length
N≥13 and odd	(N-13)/2	A string in the database encoding and (N-13)/2 bytes in length. The nul terminator is omitted.

Decoded header varint values

- First varint is 8107 (converted 135), that is the length of the record
- Second varint is 17 (converted 23) is the row id
- Third header varint(and first record varint) is record header in length 12 (converted 18)
- 1st 00 = NULL (the PK column, this will get the rowid)
- 2nd 01 = 1 Byte Integer
- 3rd 01 = 1 Byte Integer
- 4th 01 = 1 Byte Integer
- 5th 01 = 1 Byte Integer
- 6th 01 = 1 Byte Integer

Header varint values

- 7th 01 = 1 Byte Integer
- 8th 05 = 6 Byte Big Endian Integer
- 9th 01 = 1 Byte Integer
- 10th 00 = NULL
- 11th 07 = 64 bit Float
- 12th 01 = 1 Byte Integer
- 13th 25 = 12 Byte String (0x25 ->37 (37-13)/2 = 12)
- 14th 0F = 1 Byte String (0x0F ->15 (15-13)/2 = 1
- 15th 8133 = 83 Byte String (0x8133 ->179 (179-13)/2 = 83
- 16th & 17th = NULL

So what we have found so far

- A record (or row in our database) that is 135 byte long
- Consisting of the following columns:

- A row id, 5 one Byte values, a 6 Byte value, 1 Byte value, NULL, a Float, 1 Byte value, 1 Byte string, a 83 Byte sting followed by Null and Null

- Lets try to find a matching table in our database viewer!
- Close down XACT and open up the talk.sql in SQLite Expert

MICRO SYSTEMATION

Lets start building our database

- First, close down SQLite Expert
- Open up the talk.sql in XACT again
- Open up HxD hex editor
- Create a new document
- Start by copying the first 100 byte of the database header from XACT, remember to copy form the Hex window, not the ascii side
- Change the 4 Byte values at 32 and 36 and set them to 0's (this is the values related to freepages)
- Also change the 4 Byte value at 52 and 64 to 0's

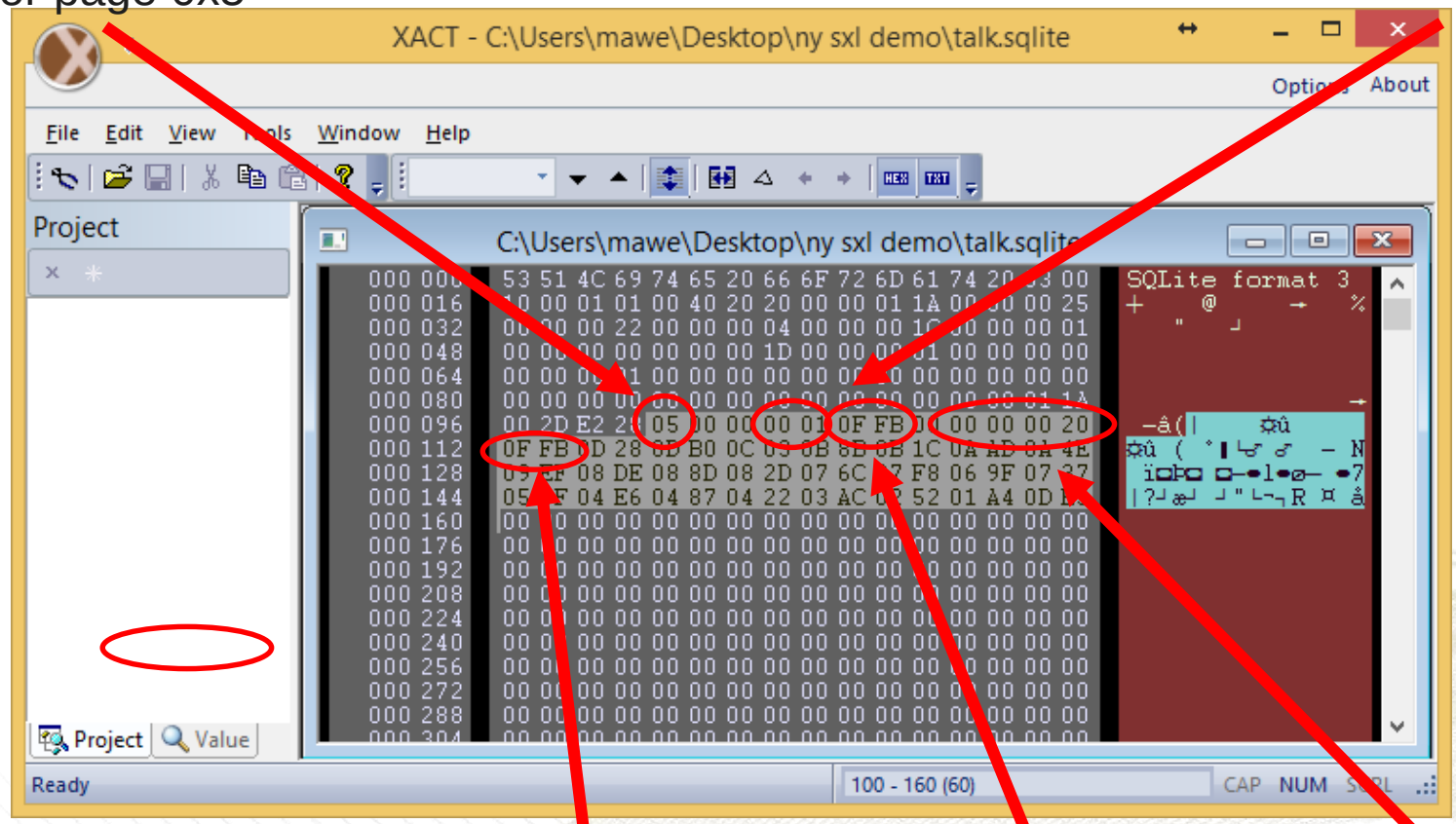
Now we have a file header

- We must now locate the database structure and paste it into our database so it knows what kind of data it will find later
- Since the Page header starts with a 05 it is an interior Page that just points to where to find the actual data
- Let’s locate where that data is located using the pointers in the database, remember where they where found?

Inside the page header

@0 1 B indicating the Page is a interior page 0x5

@5 2 B Number of Cells in this Page 1



First of the cellpointer arrays (only 1 here) 0xFFB = 4091

@7 2 B Offset to cell content area

@12 4 B (only interior Page) the rightmost Page pointer 0x20 = 32

MICRO SYSTEMATION

Digging further into the database

- So the information where to find the data of where the data is stored we have to go to offset 0F FB = 4091
- In XACT press CTRL + G and either enter the binary 4091 or hexadecimal FFB
- Now we have a 4 Byte value of the page where to find the next information (0x00 00 00 21 = 33), as well as the largest Row id on that Page (0x11 = 17)
- So to find the page address, we just have to take the Page nr – 1 * Page size again (33-1)*4096 = 131072
- So again CTRL + G in XACT and go to 131072 or 0x20000

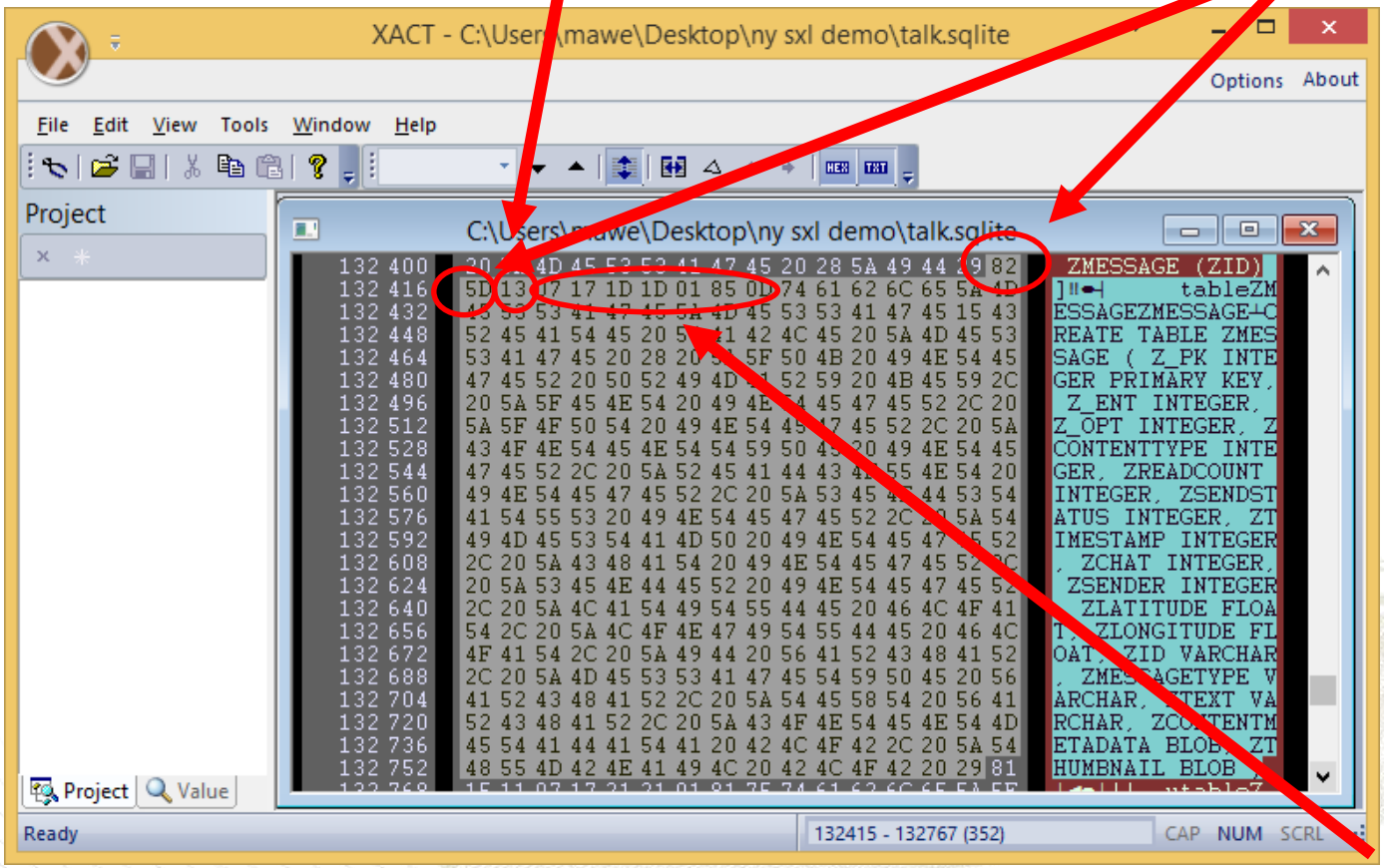
Searching for the SQL Master table

- Now we need to find the “Design and description” of the Database
- We know what table that we are looking for, ZMESSAGE table in the database
- We can search for ZMESSAGE but the will give us a lot of false positives
- So we can search for “tableZMESSAGE” that is used in the master record for the ZMESSAGE table
- In XACT standing on 131072 press CTRL – F and search for tableZMESSAGE

The record header

Row ID 0x13 = 19

This is the record size varint (82 0D = 349)



Header for the content 07 17 1D 1D 01 85 0D

MICRO SYSTEMATION

Table header

- The header values can be interpreted as follows:
- 07 = header length in bytes
- 17 = 5 Byte String
- 1D = 8 Byte String
- 1D = 8 Byte String
- 01 = 1 Byte Int
- 85 0D = varint record length
  85 0D = 28D = 653
  (653-13)/2=320 Byte String

```
CREATE TABLE sqlite_master(
  type text,
  name text,
  tbl_name text,
  rootpage integer,
  sql text
);
```

Copying the create statement we need

- So now we can copy the create statement and paste it into our own database
- Copy the whole record 349 bytes plus the varint for the record size and the row id total 352 Bytes, on the hex side
- Create a new document in HxD and paste the the data into that temporary document

Creating our page header

- Now we need to create the first Page header for our own database
- Copy the first 8 Byte of the first page header in our talk.sql that is open in XACT (byte 100 to 108)
- The data should be 05 00 00 00 01 0F FB 00
- This is the first 8 byte of the total 12 byte page header

Modifying the header

- Since the Schema table (the information we put in the temporary file in HxD) is only one record that will fit on one Page, we can use one Leaf page to hold it all
- First we need to change the page header to 0D since it is going to be a Leaf Page, not an Interior Page
- Since we are changing the Page from interior to Leaf, we didn’t need the last 4 Bytes of the Interior Page header
- Now we need to calculate where we are going to store our record (Cell with data)

Modifying the header

- Directly following the 8 byte page header are the 2 byte cell pointers
- They point to the first byte of the cell, that is written from the bottom and up
- We are only going to have one cell pointer, the one for the create statement we copied earlier

Creating pointers after the header

- Remember that the Cells where the actual payload was stored, stores from the end of the Page.
- So Page size – Cell size = offset
- 4096 – 352 = 3744 = 0E A0
- So now we need to add a pointer to that address in the Cell pointer array, that directly follows the Page header
- We also need to change the offset to the first byte of data in the cell content area (Byte 5 and 6 in the Page header)
- Header should now look like this

  0D 00 00 00 01 0E A0 00 0E A0

Modifying the header

- Now we need to modify the database Page to get the right size of it, 4096 bytes
- We do now have a 100 Byte long file header, a 8 Byte long Page header, 2 Byte long cell pointer array and we know the payload in the Cell area is going to be 352 bytes, how much padding is needed?
- 10 + 352 +100 = 462
- 4096 – 462 = 3634
- In HxD go to “Edit” and “Insert Bytes”
- Add 3634 in “Byte count” and “00” in fill pattern

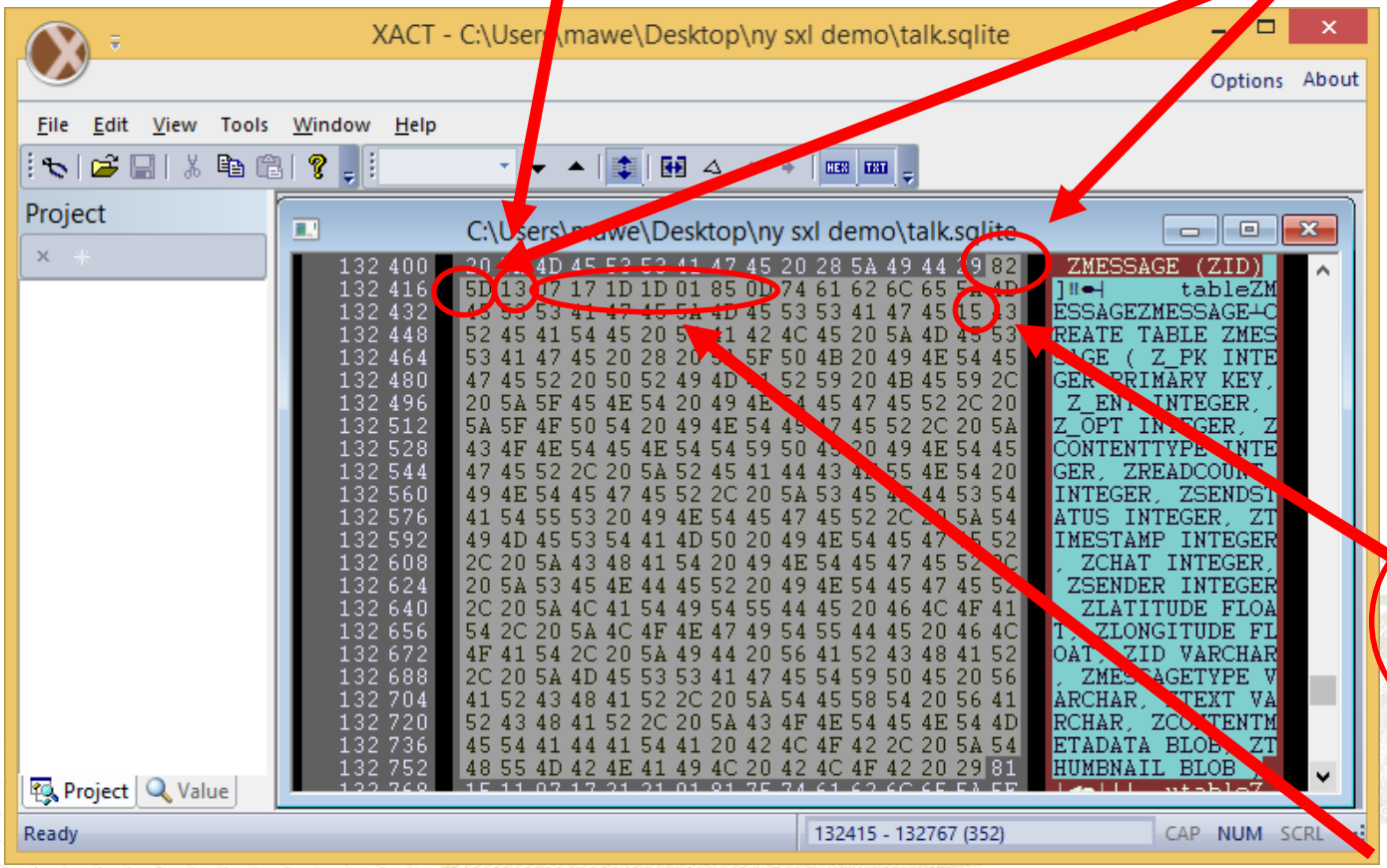
Adding the cell data

- Now, we are going to paste the actual Cell content from the temporary file in HxD to our own database file
- Go to the second tab on HxD and click in the hex value field, and press CTRL + A, CRTL +C and go back to our own database file and press CTRL + V, make sure you have the cursor after the last set of 00’s in the end of the file.
- The file should now have a total file size of 4096
- Just a few more modifications remains on our pasted data to make the first Page complete

The record header

Row ID 0x13 = 19

This is the record size varint (82 0D = 349)



Root Page integer

Header for the content 07 17 1D 1D 01 85 0D

MICRO SYSTEMATION

Modifying the cell data

- First change the Row ID from 19 to 1 (0x13 -> 01)
- Where do we find the row id?
- It was the 3rd Byte on the information we last pasted in into our file, it is found at offset 3746
- Change that 13 to 01
- Then we need to change the root Page pointer
- Page number 2 is going to be our root Page
- Once again it was the last pasted information, but a bit further down, see next page, but offset to the value is 3775
- Change the 0x15 to 0x02

Creating our second page

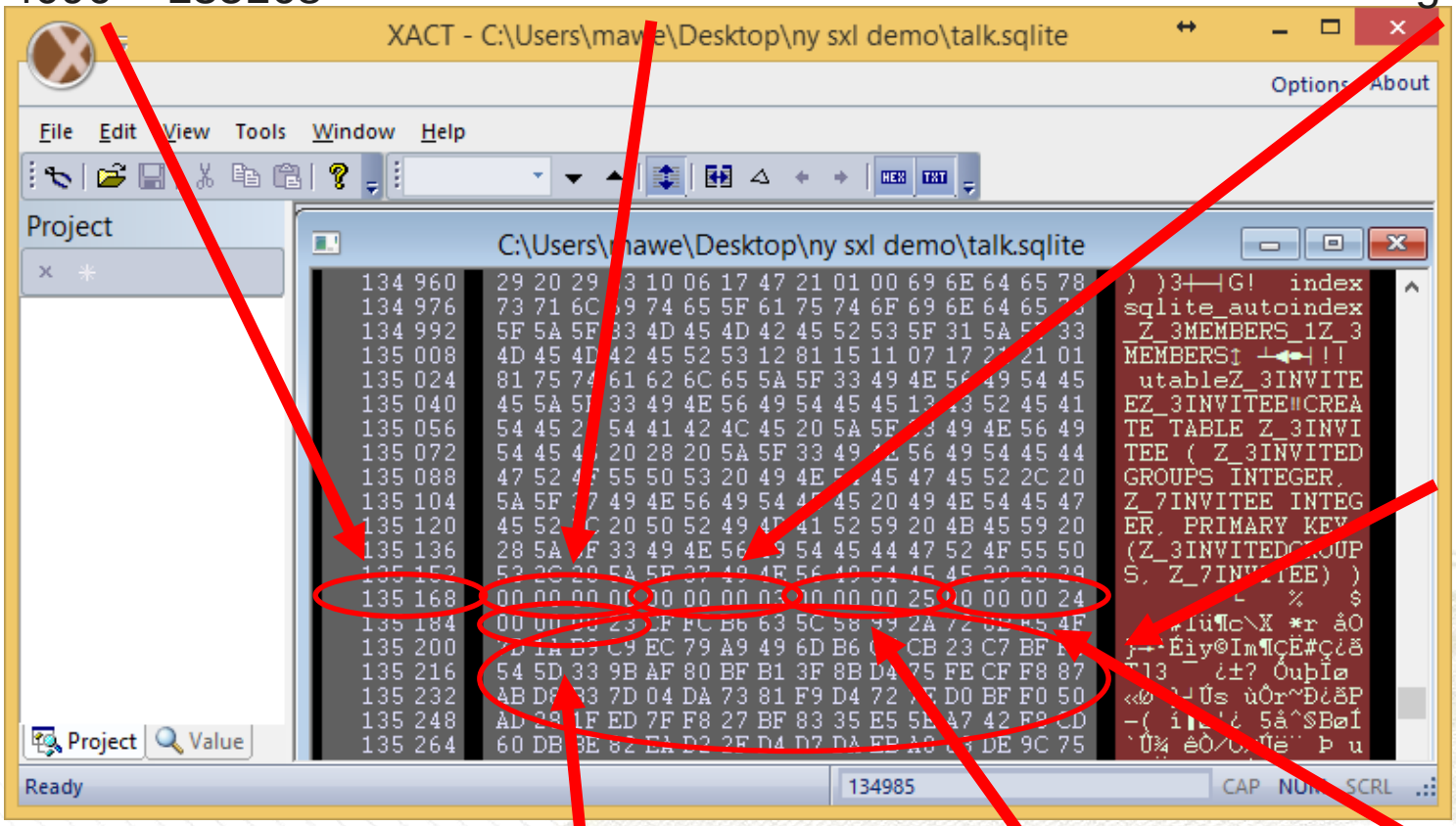
- We now have to create our second Page
- Start by cutting the 12 Bytes of the first Page header in the original database that we have open in XACT
- Paste it in at the end of our own database file, at 4096
- Since we had more than one freepage (we had 3, remember?) we need to create a interior page for handling them all (pointing them out in the database)

The freelist trunk page header

The address found in the header of the file
(34-1)*4096 = 135168

@0 4 B reference to next freelist trunk Page 0x00 00 00 00 (no more list)

@4 4 B number of freelist pointers on this Page 0x3 = 3



Overflow data = SQLite have reused an earlier page for storing the freelist trunk data

3rd freelist Page pointer 0x23 = 35

1st freelist Page pointer 0x25 = 37

2nd freelist Page pointer 0x24 = 36

MICRO SYSTEMATION

Analysing the freelist data, remember?

- We now have to open up the freelists and analyse the data that they contain
- The freelist trunkpage is Page 0x22 = 34
- The freelist's are stored on 0x23, 0x24, and 0x25
- So the freelist trunkPage is found on 135 168
- The freelist pages are found on 139 264, 143 360 and 147 456
- If we analyse the headers of those freelists, se can get the number of pointers to cell data and the actual pointers

Analysing the freelist data

- All record are ordered on the Page from the lowest number to the highest number
- The Page with the lowest records (row's) are pointed out first in the interior page and the right-most child pointer is pointing to the Page with the highest record (row)
- If the payload data can't fit in one Page, there will be a 4 Byte pointer in the end of the cell with a reference to the next Page with the remaining data, called overflow Page
- The first 4 bytes of the overflow page reference to the next Page in the overflow chain, if set to all 00's it is the last / only overflow page

Analysing the freelist data

- Just to save us some time have I already looked into the data in the freelists and they contains
- The Page 0x23 hold records between 0x01 – 0x17
- The Page 0x24 hold records between 0x18 – 0x26
- The Page 0x25 hold records between 0x17 – 0x30
- What's more important is that the record 0x18 holds a overflow chain to Page 0x22
- Where have we seen references to Page 22 earlier?

Modifying our 2nd Page header

- We now know we have 3 Leaf pages and one overflow Page
- So we can modify our header with the corresponding data

B-tree Page Header Format

Offset	Size	Description
0	1	A flag indicating the b-tree page type A value of 2 means the page is an interior index b-tree page. A value of 5 means the page is an interior table b-tree page. A value of 10 means the page is a leaf index b-tree page. A value of 13 means the page is a leaf table b-tree page. Any other value for the b-tree page type is an error.
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 the cell content area. A zero value is used to represent an offset of 65536, which occurs on an empty root page when using a 65536-byte page size.
7	1	Number of fragmented free bytes within the cell content area
8	4	The right-most pointer (interior b-tree pages only)

Modifying the header values

- So our second header before we start modifying it is now:

05000000010FFB0000000020

- We first need to change the number of cells on this Page
- Then we need to change the right most pointer
- And then we need to change the values where the cell data starts
- And we need to add cell pointers to the cells containing the data
- The data in the cells will just be a 4 byte Page number, and the biggest row id we will find on that Page

Modifying the header values

- So there will be 2 five Bytes cells in the end of the Page
- That means the first cell is going to be 4096-5=4091=0FFD
- The second cell is going to be 4091-5=4086=0FF6
- We don’t need to point out the last freepage since that is done by pointing it out in the 4 byte right-most pointer

Total number of cells on this page

First page pointer 0xFFF-5

- And after we applied the changes:
- 05 00 00 00 02 0F F6 00 00 00 00 05 0F FB 0F F6

Address to first free byte

Page number of the rightmost page (last page)

Second page pointer 0xFFF-5

MICRO SYSTEMATION

Finishing up our second page

- We now have to pad the rest of the file with 00's so all we have to do is take the Page size of 4096 and remove the 12+2+2 Byte for header and pointers and also deduct 5+5 Byte for the cell data in the end of the page = 4070 Bytes
- So in HxD click "Edit" and "Insert Bytes" and add 4070 bytes of 00's to our database file
- Then we need to add the cell data, 4 Bytes for the Page number and 1 byte biggest row on that Page, repeated twice
- The Page numbers will be 3 and 4, we will have a total of 6 pages (start page, and this the second page, 3 freelist and 1 overflow Page)

Adding the cell content

- So now, lets populate our second cell with the mentioned values
- Since data is written from the back of the cell and upwards we will for efficiency write the values in wrong order
- Add 00 00 00 04 for the page number for second content page
- Add 26 for the highest row on that page
- Add 00 00 00 03 for the first content page
- And finally add 17 for the highest row on that page

Pasting in our freelist data Pages

- Now we can paste in the Leaf Pages
- So the data from page 0x23-0x25 can just be copied and pasted from XACT to our database
- So start is 0x23 = 35 (35-1) * 4096=139264
- End is 0x25 = 37 * 4096 = 151552
- Pages do not need to be stored in order, just the cells in the pages, but now they are conveniently in order so lets keep it so
- Now we need to get the overflow Page in as well… that was the Page that SQLite has reused as freelist trunk Page

Pasting in the overflow page

- Now we can cut and paste the overflow Page as the last Page in our database
- So go to start of page 0x22 in xact (135168) and mark and scroll to the last byte of that page (139264) and copy and paste it in in our database
- Finally we need to change the pointer to the overflow page in the row 18 on page 4 to the new page number 6

Last thing to do:

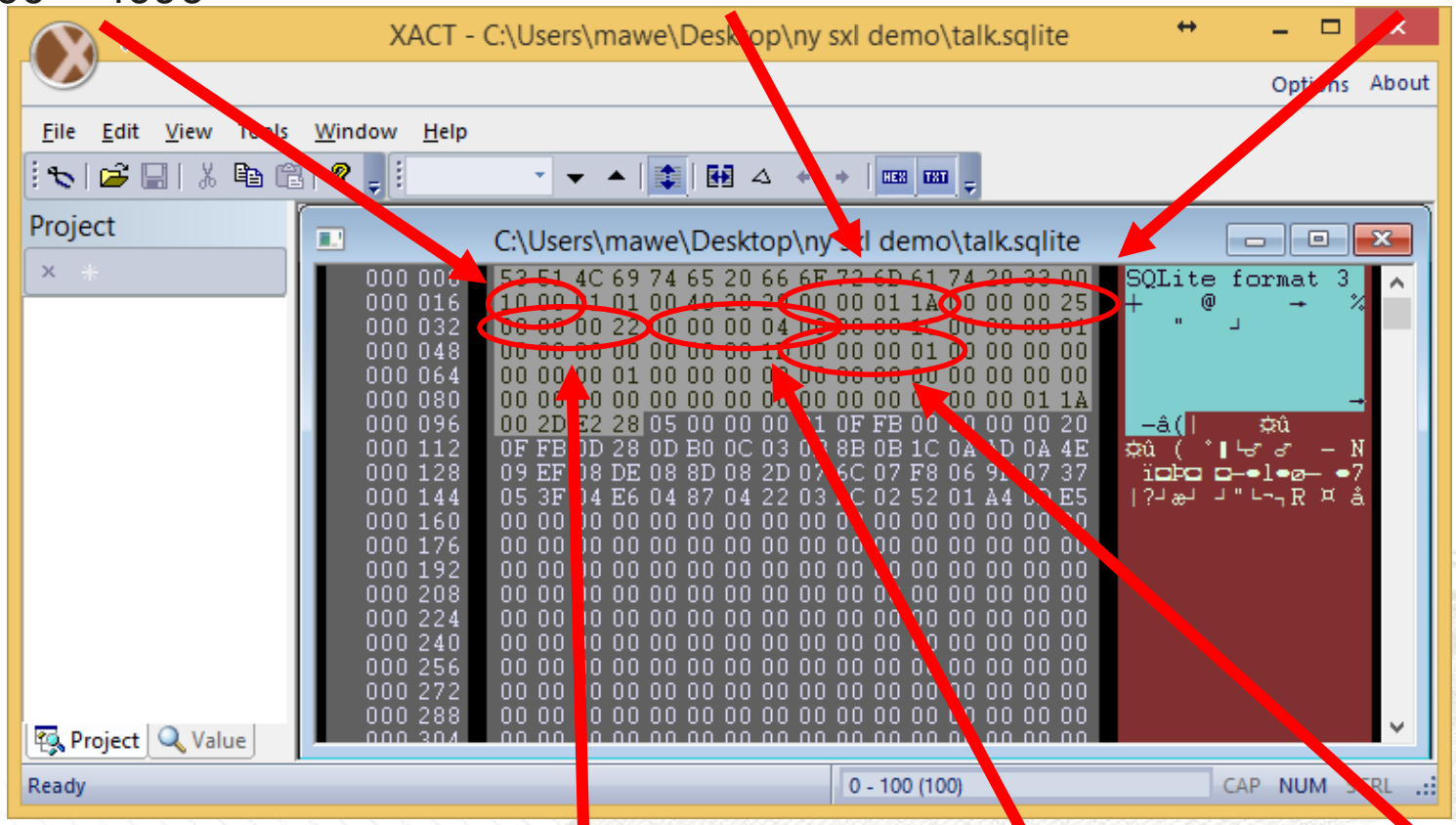
- In HxD click on “Search” and “Find” and look for 00000022 (the reference in to the old place for the overflow page)
- We can easily detect which is the right one to change (2nd hit) because it’s address is on the end of the page
- Change the 22 to 06
- And finally we need to set the total size of the database
- Change the 4 Byte in the file header at offset 28 over the total amount of Pages in the file which is now 6 and not 37 (0x25)

Inside the file header

@16 2 B indicating the Page size 0x1000 = 4096

@24 4 B counter how many times data has changed in the file 0x11A = 282

@28 4 B How many Pages the database consist of 0x25 = 37



@32 4 B Where to find the freelist trunk Page 0x22 = 34

@36 4 B How many freelists the database contains 4

@56 4 B The encoding scheme used 1 means UTF8

MICRO SYSTEMATION

Last thing to do:

- We are DONE! ☺

- Save the file with a .sql file extension and open it with SQLite Expert Personal
- Reap the fruits of our labour and browse through the deleted messages from Line messenger

- Also take a look at row 24 (remember the page 0x18 with a overflow page, that is actually the pic we se under thumbnail eventhough some bytes where overwritten)

Conclusions

- Apps today are storing data in SQLite3 databases
- Most apps are not encrypting their data (yet….)
- Helper applications can be used to search for evidential traces left by system and user apps
- Rebuilding databases are hard work, but can be done

- And finally the sales pitch… with XRY's intelligent SQLite parser, this is all done automatically… and the deleted information ends up in the report without any user interaction…

MICRO SYSTEMATION

MICRO SYSTEMATION