### 数据库文件的结构

<https://blog.csdn.net/popvip44/article/details/53056949>

主数据库文件以.db为后缀。数据库文件内部划分为若干个页。页的大小可以是512到65536之间任意的2次方中的一个，但是同一个数据库中所有页的大小是相同的。一个数据库文件最多拥有2147483646(2^31-2)个页。

页的编号从1开始，所以数据库文件偏移为0处开始到页大小值前一字节处的页页号为1。紧跟在页面1后面的页面按递增顺序依次指定页号。

可以使用工具SQlite Page Explorer查看数据库文件里面的页信息，工具在github的地址为<https://github.com/rubydongle/sqlite3_page_explorer>。

页号为1的页和其它的页有一些区别，其前100个字节中存储着数据库文件头信息。即数据库文件头100个字节中存储着该数据库文件的文件头信息。

第1页（master catalog）存储了所有的表，由于master catalog也是通过b+树组织的，而本例中只有一个表，第1页没有占满，所以第1页是master catalog所代表的b+树的叶子节点，也是master catalog所代表的b+树的唯一节点。

数据库文件头中包含下面信息：

|  |  |  |
| --- | --- | --- |
| 偏移(Offset) | 大小(Size) | 描述(Description) |
| 0 | 16 | 文件头字符串”SQLite format 3\000”。 The header string: “SQLite format 3\000” |
| 16 | 2 | 页大小每个页空间大小，单位byte.必须是512~32768之间2^xxx,如果是1表示页大小为65536 |
| 18 | 1 | 文件写格式。1表示传统格式（Legacy）2表示WAL。File format write version. 1 for legacy; 2 for WAL. |
| 19 | 1 | 文件读格式。1表示传统格式（Legacy）2表示WAL。File format read version. 1 for legacy; 2 for WAL. |
| 20 | 1 | 每页尾部保留未使用空间数，一般为0。Bytes of unused “reserved” space at the end of each page. Usually 0. |
| 21 | 1 | 最大嵌入负荷片段。内部结点页的最大嵌入负载(嵌入负载是个百分数，0~255代表0%到100%)，这个值是用来限制最大的单个record的，如果最大嵌入负载是25%，那么单条record就不能超过1/4页，超过的部分会转移到溢出页。Maximum embedded payload fraction. Must be 64. |
| 22 | 1 | 最小嵌入负荷片段。内部结点页的最小嵌入负载。Minimum embedded payload fraction. Must be 32. |
| 23 | 1 | 叶负荷片段。叶子结点页的最小嵌入负载，叶子节点页的最大负载永远是100%，所以只给出叶子节点页的最小负载限制Leaf payload fraction. Must be 32. |
| 24 | 4 | 文件修改次数。文件被修改的次数，本例中该值是2，创建table和插入数据共修改了两次文件。File change counter. |
| 28 | 4 | 数据库文件页面总数。Size of the database file in pages. The “in-header database size”. |
| 32 | 4 | 第一个Freelist Trunk page的页号。自由页链表头。Page number of the first freelist trunk page. |
| 36 | 4 | FreeList page页的总数。自由页页数。Total number of freelist pages. |
| 40 | 4 | The schema cookie.当数据库schema改变时该值就会加1,， |
| 44 | 4 | schema格式。The schema format number. Supported schema formats are 1, 2, 3, and 4. |
| 48 | 4 | 默认页缓存大小。Default page cache size. |
| 52 | 4 | 在自动清空模式或者增量清空模式下，这个值是数据库文件中最大的B树根节点页。在其他模式下，这个值是0.The page number of the largest root b-tree page when in auto-vacuum or incremental-vacuum modes, or zero otherwise. |
| 56 | 4 | 数据库文本编码格式。1表示UTF-8 2表示UTF-16le 3表示UTF-16be。The database text encoding. A value of 1 means UTF-8. A value of 2 means UTF-16le. A value of 3 means UTF-16be. |
| 60 | 4 | 通过PRAGMA user\_version设置的“用户版本号”。The “user version” as read and set by the user\_version pragma. |
| 64 | 4 | 非零表示增量清空模式。零表示其他模式。True (non-zero) for incremental-vacuum mode. False (zero) otherwise. |
| 68 | 4 | 应用程序号,用于表明sqlite属于特定的应用程序。通过命令PRAGMA application\_id设置的”Application ID” |
| 72 | 20 | 保留字节。保留用作后续扩展，必须是0. |
| 92 | 4 | The version-valid-for number. |
| 96 | 4 | SQLite 版本号 |

SQLite数据库文件中的页有下列类型：

* B-树页(A b-tree page)
* 有效负荷溢出页(A payload overflow page)
* 空闲列表页(A freelist page)
* (A pointer map page)
* (The lock-byte page)

### B-树页

B-树页用来存储B-树信息，包括数据库表信息和数据库索引信息。B-树页分为下面四类

1. A table b-tree interior page
2. A table b-tree leaf page
3. An index b-tree interior page
4. An index b-tree leaf page

The b-tree page header is 8 bytes in size for leaf pages and 12 bytes for interior pages. All multibyte values in the page header are big-endian. The b-tree page header is composed of the following fields:

|  |  |  |
| --- | --- | --- |
| Offset | Size | Description |
| 0 | 1 | The one-byte flag at offset 0 indicating the b-tree page type.   * A value of 2 (0x02) means the page is an interior index b-tree page. * A value of 5 (0x05) means the page is an interior table b-tree page. * A value of 10 (0x0a) means the page is a leaf index b-tree page. * A value of 13 (0x0d) means the page is a leaf table b-tree page.   Any other value for the b-tree page type is an error. |
| 1 | 2 | The two-byte integer at offset 1 gives the start of the first freeblock on the page, or is zero if there are no freeblocks. |
| 3 | 2 | The two-byte integer at offset 3 gives the number of cells on the page. |
| 5 | 2 | The two-byte integer at offset 5 designates the start of the cell content area. A zero value for this integer is interpreted as 65536. |
| 7 | 1 | The one-byte integer at offset 7 gives the number of fragmented free bytes within the cell content area. |
| 8 | 4 | The four-byte page number at offset 8 is the right-most pointer. This value appears in the header of interior b-tree pages only and is omitted from all other pages. |

B-树页头

0x64: b树节点页类型。0x0d表示表的b+树的叶子节点页。

0x65~0x66: 第一个自由块的位置，自由块通过链表链接起来。自由块通常是由于cell的删除或更新产生的，自由块缩小到一定程度就成为了碎片，sqlite会按照一定的逻辑判断碎片数量并进行碎片整理。

0x67~0x68: 本页中cell的数量。本例中是1，这个cell存储了表t1。

0x69~0x6a: 第一个cell的偏移量。本例中是0xfd0 + 0。

0x6b: 碎片空间总大小。接下来的4bytes应该是最右孩子节点。但是这是叶子节点页，没有孩子，所以这4bytes省略了。

0x6c~0x6d: 指出cell的偏移量。具体参考官方文档中cell和cell pointer的存储方式。

Cell的格式

不同的B-树Cell的格式是不一样的。

Table B-Tree Leaf Cell (header 0x0d):

A varint which is the total number of bytes of payload, including any overflow

A varint which is the integer key, a.k.a. "rowid"

The initial portion of the payload that does not spill to overflow pages.

A 4-byte big-endian integer page number for the first page of the overflow page list - omitted if all payload fits on the b-tree page.

Table B-Tree Interior Cell (header 0x05):

A 4-byte big-endian page number which is the left child pointer.

A varint which is the integer key

Index B-Tree Leaf Cell (header 0x0a):

A varint which is the total number of bytes of key payload, including any overflow

The initial portion of the payload that does not spill to overflow pages.

A 4-byte big-endian integer page number for the first page of the overflow page list - omitted if all payload fits on the b-tree page.

Index B-Tree Interior Cell (header 0x02):

A 4-byte big-endian page number which is the left child pointer.

A varint which is the total number of bytes of key payload, including any overflow

The initial portion of the payload that does not spill to overflow pages.

A 4-byte big-endian integer page number for the first page of the overflow page list - omitted if all payload fits on the b-tree page.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Datatype | Appears in... | | | | Description |
| **Table Leaf (0x0d)** | **Table Interior (0x05)** | **Index Leaf (0x0a)** | **Index Interior (0x02)** |
| 4-byte integer |  | ✔ |  | ✔ | Page number of left child |
| varint | ✔ |  | ✔ | ✔ | Number of bytes of payload |
| varint | ✔ | ✔ |  |  | Rowid |
| byte array | ✔ |  | ✔ | ✔ | Payload |
| 4-byte integer | ✔ |  | ✔ | ✔ | Page number of first overflow page |

0xfd0: 用变长度方式存储的数据大小，本例中是46，即从0xfd2到0xfff共46bytes数据。关于变长度方式存储，参见我的另一篇文章 SQLite 变长度整型(varint)编码解码方法 。

0xfd1: 键值大小，如果键值是整型变量，那么这个值就是键值本身。本例中sqlite为表t1自动分配了一个键值01。

0xfd2: header size。本例中是06，即0xfd3到0xfd7都是数据类型。数据类型的具体定义见官方文档。

0xfd3: 17转化为十进制就是23,(23-13)/2 = 5,代表第一个数据是5个字节大小的text，即“table”。

0xfd4: 11转化为十进制就是17，(17-13)/2 = 2,即t1。

0xfd5: 同上。

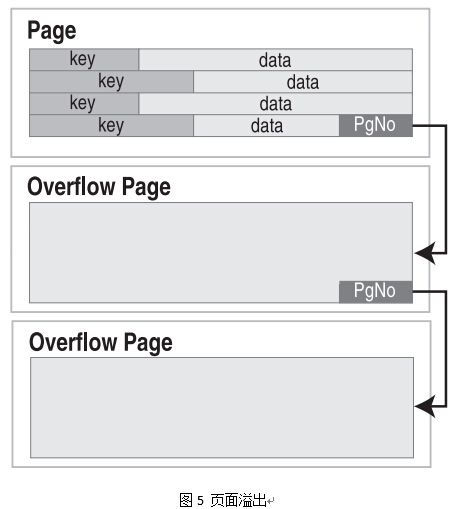
0xfd6: 以二进制补码形式存储的整型数据所占的byte。对应0xfe1的02，02是1byte。

0xfd7: 49转化为十进制就是73,(73-13)/2 = 30,即0xfe2到0xfff，刚好30个bytes。

### 有效负荷溢出页(A payload overflow page)

有效载荷及其内容可有不同的大小。然而，页面大小是固定不变的。因此，给定的有效载荷总有可能超出单页装载大小。这种情况发生时，额外的有效载荷将添加到溢出页面的链接链表上。由此看来，有效载荷将在有序的链接链表中显示，如图所示。

图中第4个有效载荷超出了当前页所能装载的大小。因此，B-tree模块创建了溢出页来装载。实际上，一个溢出页也不能装载，因此，又链接了第二个溢出页。这实际上就是处理二进制大对象的方法。使用真正的大字段时，最后都采用页链接链表来存储。如果blob字段太大，这种方式效率很低，此时，可考虑创建外部文件来存储blob数据，并将外部文件名保存在记录中。



### 空闲列表页(Freelist Page)

空闲列表页有两种

1. A freelist trunk page
2. A freelist leaf page

A database file might contain one or more pages that are not in active use. Unused pages can come about, for example, when information is deleted from the database. Unused pages are stored on the freelist and are reused when additional pages are required.

The freelist is organized as a linked list of freelist trunk pages with each trunk page containing page numbers for zero or more freelist leaf pages.

A freelist trunk page consists of an array of 4-byte big-endian integers. The size of the array is as many integers as will fit in the usable space of a page. The minimum usable space is 480 bytes so the array will always be at least 120 entries in length. The first integer on a freelist trunk page is the page number of the next freelist trunk page in the list or zero if this is the last freelist trunk page. The second integer on a freelist trunk page is the number of leaf page pointers to follow. Call the second integer on a freelist trunk page L. If L is greater than zero then integers with array indexes between 2 and L+1 inclusive contain page numbers for freelist leaf pages.

Freelist leaf pages contain no information. SQLite avoids reading or writing freelist leaf pages in order to reduce disk I/O.

The number of freelist pages is stored as a 4-byte big-endian integer in the database header at an offset of 36 from the beginning of the file. The database header also stores the page number of the first freelist trunk page as a 4-byte big-endian integer at an offset of 32 from the beginning of the file.

### Pointer Map or Ptrmap Pages

Pointer map or ptrmap pages are extra pages inserted into the database to make the operation of auto\_vacuum and incremental\_vacuum modes more efficient. Other page types in the database typically have pointers from parent to child. For example, an interior b-tree page contains pointers to its child b-tree pages and an overflow chain has a pointer from earlier to later links in the chain. A ptrmap page contains linkage information going in the opposite direction, from child to parent.

### The Lock-Byte Page

The lock-byte page is the single page of the database file that contains the bytes at offsets between 1073741824 and 1073742335, inclusive. A database file that is less than or equal to 1073741824 bytes in size contains no lock-byte page. A database file larger than 1073741824 contains exactly one lock-byte page.

The lock-byte page is set aside for use by the operating-system specific VFS implementation in implementing the database file locking primitives. SQLite does not use the lock-byte page. The SQLite core will never read or write the lock-byte page, though operating-system specific VFS implementations may choose to read or write bytes on the lock-byte page according to the needs and proclivities of the underlying system. The unix and win32 VFS implementations that come built into SQLite do not write to the lock-byte page, but third-party VFS implementations for other operating systems might.

The lock-byte page arose from the need to support Win95 which was the predominant operating system when this file format was designed and which only supported mandatory file locking. All modern operating systems that we know of support advisory file locking, and so the lock-byte page is not really needed any more, but is retained for backwards compatibility.