

Firebird のヘッダを読み書きする

林 務

2006/03/21

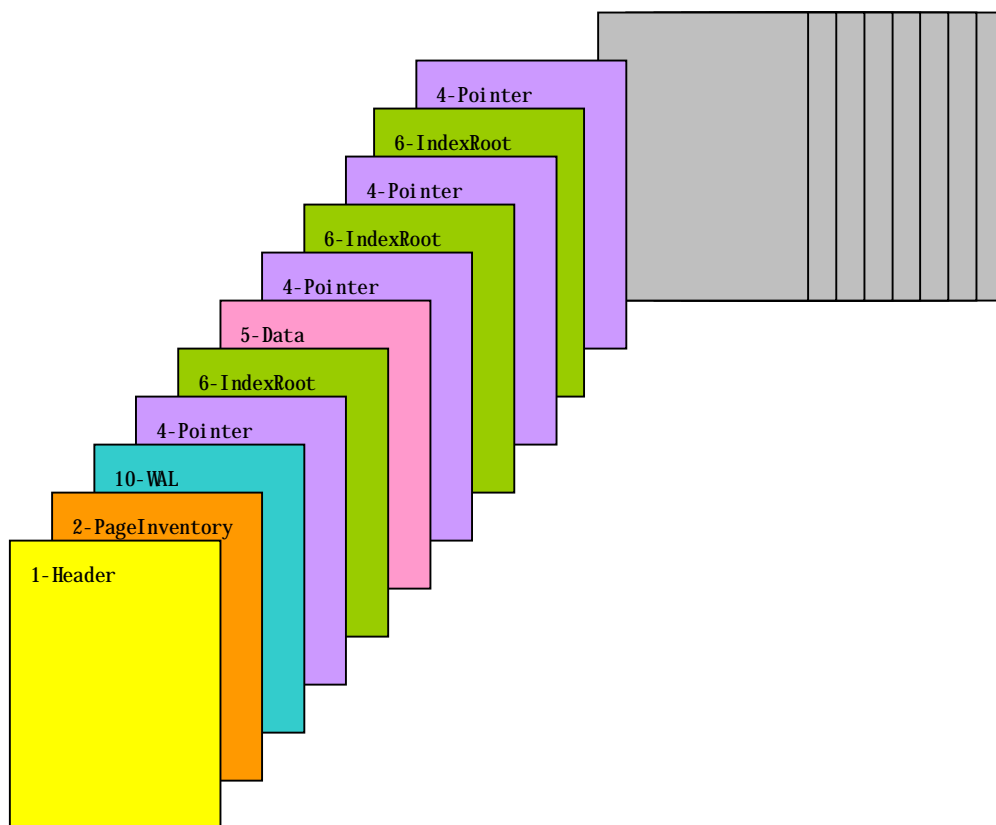
Delphi には、ボーランド社製リレーショナルデータベース **InterBase** 用のコンポーネント **InterBaseExpress (IBX)** が付属しています。**InterBase** のローカルライセンス版も付属しているため、多くの方が一度は使ったことがあるのではないかと思います。**Firebird** はご存じの通り **InterBase** から派生したオープンソースのリレーショナルデータベースです。当然、最初のバージョンであるバージョン **1.0** 系列は **IBX** で問題なくアクセスすることが出来ます。

Firebird に接続し、データを読み書きするためには **Firebird** の **API (Application Programming Interface)** を利用して、クライアントから **TCP/IP** 等のネットワークプロトコルを使用して **Firebird** サーバと通信をする必要があります。**Access** 等のファイルベースのデータベースとは違い、データベースファイルにアクセスするのはサーバプロセスのみとなり、そのため高度な排他制御と並行処理を可能としています。

とはいえ、データベースファイルそのものは単なるバイナリファイルにすぎません。手元に置けば直接読み書きすることも可能です。**Firebird** のデータベースファイルの形式は **ODS (On Disk Structure)** と呼ばれ、現在の正式リリースであるバージョン **1.5** 系列では **ODS 10.1** が使用されています。今回はこの **Firebird** のヘッダをデータベースファイルから直接読み書きしてみます。

Page types

Firebird のデータベースは、1024・2048・4096・8192・16384 バイトの大きさのページで構成されています。それぞれのページは、それぞれ特定の目的で使用され、その Page type は ①データベースヘッダーページ ②ページインベントリページ ③トランザクションインベントリページ ④ポインターページ ⑤データページ ⑥インデックスルートページ ⑦インデックスページ ⑧ブロブデータページ ⑨ジェネレーターページ ⑩ログインフォページの 10 種類があります。ライトアヘッドログに関する機能は現在は使用されていませんが、ページ自体は 3 番目に必ず存在しています。



ods.h の定義を以下に示します。

```
/* Page types */
```

```
const SCHAR pag_undefined           = 0;
const SCHAR pag_header               = 1;           /* Database header page */
const SCHAR pag_pages               = 2;           /* Page inventory page */
const SCHAR pag_transactions         = 3;           /* Transaction inventory page */
const SCHAR pag_pointer              = 4;           /* Pointer page */
```

```
const SCHAR pag_data          = 5;          /* Data page */
const SCHAR pag_root         = 6;          /* Index root page */
const SCHAR pag_index        = 7;          /* Index (B-tree) page */
const SCHAR pag_blob         = 8;          /* Blob data page */
const SCHAR pag_ids           = 9;          /* Gen-ids */
const SCHAR pag_log           = 10;         // Write ahead log information
DEPRECATED
const SCHAR pag_max           = 10;         /* Max page type */
```

Basic page header

さて、それぞれのページには先頭に 16 バイトの **Basic page header** が配置されています。先頭の 1 バイトは上で挙げたページタイプ、2 バイト目はページ **Flag** と呼ばれ、現在の所 **Blob** ページと **Index** ページ(**b-tree**)でしか使用されていません。ヘッダーページ等では 0 が入っています。

続く 2 バイトはチェックサムと表記されていますが、現状では常に 10 進数で 12345 が入っています。16 進数では **0x3039** となりますが、ファイル上のバイトオーダーはリトルエンディアンになっているので **0x3930** と記録されています (以下同様)。

その次の 4 バイトはページの **Generation** (世代) になります。これは、各ページに書き込みが行われる度にインクリメントされます。作製したばかりのデータベースファイルのヘッダページの **Generation** は **0x04000000** となっています。

最後の 8 バイトは、**WriteAheadLog** 用のシーケンスナンバーとオフセットという事になっていますが、前述したように **WAL** は現在のところ利用されていないため、この 8 バイトも使用されていません。

Page Type	Page Flags	Check Sum		Gene ration				Page SCN				Offset				
-----------	------------	-----------	--	-------------	--	--	--	----------	--	--	--	--------	--	--	--	--

ods.h の定義を以下に示します。

```
/* Basic page header */
```

```
struct pag
```

```
{
```

```
    SCHAR pag_type;
```

```
    SCHAR pag_flags;
```

```
    USHORT pag_checksum;
```

```
    ULONG pag_generation;
```

```
    // We renamed pag_seqno for SCN number usage to avoid major ODS version bump
```

```
    ULONG pag_scn;                /* WAL seqno of last update */
```

```
    ULONG reserved;              /* Was used for WAL */
```

```
};
```

Header Page

各データベースファイルに必ずヘッダーページが存在し、これは必ず先頭ページにおかれることになっています。最初のファイルのヘッダーページには、データベースのページサイズや次のトランザクション ID、その他の設定等々が書き込まれています。二次ファイル以降のヘッダーページにはファイルのサイズと、自分の次のファイル名のみが書き込まれています。

以下に、Firebird1.5 で生成したページサイズ 4096 のデータベースファイルのヘッダーを示します。

0x00000000 ...												0x0000000F			
01	00	39	30	04	00	00	00	00	00	00	00	00	00	00	00
pageheader															
00	10	0A	00	03	00	00	00	00	00	00	00	01	00	00	00
page_size		ods_version		PAGES				next_page				oldest_transaction			
02	00	00	00	03	00	00	00	00	00	02	01	BD	D1	00	00
oldest_active				next_transaction				sequence		flags		creation_date[2]			
60	D0	4A	25	00	00	00	00	00	00	00	00	10	00	01	00
				attachment_id				shadow_count				implementation		ods_minor	
01	00	60	00	00	00	00	00	01	00	00	00	02	00	00	00
ods_minor_original		end		page_buffers				bumped_transaction				oldest_snapshot			
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
misc[4]															
00	00	00	00												
data[1]															
0x00000060 ...												0x0000006F			

ods.h の定義を以下に示します。

```
/* Header page */
```

```
typedef struct hdr {
    struct pag_hdr_header;
    USHORT hdr_page_size;           /* Page size of database */
    USHORT hdr_ods_version;        /* Version of on-disk structure */
    SLONG hdr_PAGES;               /* Page number of PAGES relation */
    ULONG hdr_next_page;          /* Page number of next hdr page */
    SLONG hdr_oldest_transaction; /* Oldest interesting transaction */
    SLONG hdr_oldest_active;      /* Oldest transaction thought active */
};
```

```
SLONG hdr_next_transaction;    /* Next transaction id */
USHORT hdr_sequence;          /* sequence number of file */
USHORT hdr_flags;             /* Flag settings, see below */
SLONG hdr_creation_date[2];   /* Date/time of creation */
SLONG hdr_attachment_id;      /* Next attachment id */
SLONG hdr_shadow_count;       /* Event count for shadow synchronization */
SSHORT hdr_implementation;    /* Implementation number */
USHORT hdr_ods_minor;         /* Update version of ODS */
USHORT hdr_ods_minor_original; /* Update version of ODS at creation */
USHORT hdr_end;               /* offset of HDR_end in page */
ULONG hdr_page_buffers;       /* Page buffers for database cache */
SLONG hdr_bumped_transaction; /* Bumped transaction id for log optimization */
SLONG hdr_oldest_snapshot;    /* Oldest snapshot of active transactions */
SLONG hdr_misc[4];            /* Stuff to be named later */
UCHAR hdr_data[1];           /* Misc data */
} *HDR;
```

GLink

Glink は、チェコ共和国の **Ivan Prenosil** 氏が公開している **c** コードベースの **Firebird/InterBase** 用ユーティリティで、データベースの 2 次ファイルを配置変更したい場合に通常はバックアップ+リストアを行わなくてはならないところを、直接ページヘッダーを書き換えることでその手間を省こうというツールです。システムテーブル **RDB\$FILES** の値は書き換えないので、**IBOConsole** 等の GUI ツールで 2 次ファイルを表示させると元のままになっているように見えますが、設定自体は変更されています。

glink.exe の使用方法は以下の通りです。

```
C:\ >glink
```

```
Copyright(C)2001 Ivan Prenosil
```

```
Contributed by Tomneko in Japan
```

```
This program is intended to simplify moving multifile databases.
```

```
It changes name of next file in chain stored in database header
```

```
(by directly modifying header page of IB/FB database).
```

```
It does not rename/move the file itself.
```

```
!!! Use at your own risk !!!
```

```
Syntax: glink [switches] [db_file_name [new_name]]
```

```
-h: print header
```

```
-H: print full header
```

```
-?: print this help text
```

```
-z: print version information
```

```
glink <db_file_name>
```

```
will show name of next file in chain.
```

```
glink <db_file_name> <new_name>
```

```
will change name of next file.
```

これを **Delphi** でリライトして、多少機能を追加してみました。変更したのは、①新しく指定する 2 次データベースファイルの存在チェックと絶対パスであるかどうかのチェック、② **-H** オプションで表示されるヘッダー情報に **PAGES** と **next_page** を追加、③2 次データベースファイル以降でチェックサムエラーが表示される不具合の修正、④ヘッダーの表示を **BasicPageHeader** とその他を分けて見やすくした、等です。

ソースコードを以下に示します。

GLink.dpr, GLink_functions.pas 参照

まず、ParamCount から引数をパースするかどうかを判断しています。グローバルな変数の初期化もこの中で行っているため、引数が無くてもとりあえず呼ぶようになっています。引数に従って、グローバル変数 sw_show_version, sw_show_help, sw_show_header, sw_show_header_full が設定され、その後の動作が決定されます。

ページヘッダの内容は、get_db_header() 関数で取得され、hdr_page で宣言されたレコード型の変数 header_page に読み込まれます。この時に、まず MZ_PAGE_SIZE=1024 バイトを読み込んでから、再度 header_page.fix_data.hdr_page_size の大きさで読み込んでいます。可変長のページサイズを処理するためにこのような手続きを取っています。

show_db_header() 関数の中では、この header_page の各フィールドを書式化して表示しています。Firebird の内部時刻型は日付部分を 32 ビット整数で、時刻部分を同じく 32 ビット整数で格納しています。これを C の time_t 型に変換するために Ivan が書いたコードが以下の部分です。

```
ti = 86400 * (header_page.fix_data.hdr_creation_date[0] - 40587) + _timezone
    + (header_page.fix_data.hdr_creation_date[1] / 10000);
if (_daylight)
    ti = ti - 3600;
```

hdr_creation_date[0]から 40587 を引いた値に 86400 (一日の秒数) を乗じているのは、これが日付部分であるためですが、40587 はどこから来ているのでしょうか。いろいろと調べてみればどこかに書いてあるのですが、面倒なのでバイナリエディタで hrd_creation_date[0][1] をそれぞれ 0 で書き換えてみました。結果、1858/11/17 00:00:00 が起点となっていることがわかりました。40587 を 365 で割ると約 111 年になりますから、time_t 型の 1970/1/1 00:00:00 からの経過日数との差がこの数値になっていることがわかります。ちなみに、Firebird の TimeStamp 型自体は西暦 100 年から 32768 年まで対応しています。

Delphi では日付時刻型は整数部が 1899 年 12 月 31 日からの経過日数で、時刻部分は小数部で表現されています。そのため、以下のような変換を行いました。また、time_t 型との日付差を修正するため、Delphi に組込の UnixDateDelta 定数を利用しました。

```
ti := (header_page.fix_data.hdr_creation_date[0] - 40587);
ti := incsecond(ti, trunc(header_page.fix_data.hdr_creation_date[1] / 10000));
if (_daylight > 0) then ti := incsecond(ti, -3600);
ti := ti + UnixDateDelta;
```

本来は、ibase.h で定義されている、isc_decode_date() 関数を利用するのが本筋なのでしょうが、内部格納形式を理解する一助にはなるかと思います。もっとも、今後内部格納形式が変更さ

れた場合には、API ルーチンはそれを考慮して正しい値を返すはずですが、上述の方法では対応出来ないことは言うまでもありません。

さて、`struct hdr` の最終バイトである `UCHAR hdr_data[1]` 以降には `clumplets` と呼ばれる可変長のデータが格納されています。以下に `ods.h` の定義を示します。

```
/* Header page clumplets */

/* Data items have the format

    <type_byte> <length_byte> <data...>
*/

#define HDR_end                0
#define HDR_root_file_name     1    /* Original name of root file */
#define HDR_journal_server     2    /* Name of journal server */
#define HDR_file               3    /* Secondary file */
#define HDR_last_page          4    /* Last logical page number of file */
#define HDR_unlicensed         5    /* Count of unlicensed activity */
#define HDR_sweep_interval     6    /* Transactions between sweeps */
#define HDR_log_name           7    /* replay log name */
#define HDR_journal_file       8    /* Intermediate journal file */
#define HDR_password_file_key  9    /* Key to compare to password db */
#define HDR_backup_info        10   /* WAL backup information */
#define HDR_cache_file         11   /* Shared cache file */
#define HDR_max                11   /* Maximum HDR_clump value */
```

`clumplets` のタイプが `1, 2, 3, 7, 11` の時は文字列型なので、`length_byte` の長さに従って文字列を取り出しています。また、`8, 9, 10` は現在使用されていないため除外しています。`4, 5, 6` の場合は整数型になるのですが、Ivanさんのコードでは後ろから前へ1バイトずつ取り出して `Integer` 型に加算しては左へ8ビットシフトしていました。数値型の値が4バイトまでで入っているという仮定の下ではこれは合理的なやり方です。

```
int    i, j, k, len;

for (j = 0, k = len;
     k--;
     j = (j << 8) + header_page.var_data[i+k+2]);
```

```
printf (str, j);
```

ひとまず、そのまま Delphi のコードに置き換えてみました。

```
i, j, k, len: integer;  
  
j := 0; k := len;  
repeat  
    dec(k);  
    j := (j shl 8) + header_page.var_data[i+k+2];  
until (k = 0);  
writeln(format(str, [j]));
```

C に比較すると長くなってしまふのは致し方ないですが、もっとうまい書き方があるかも知れません。しかし、今回は数値型のデータには全て 4 バイトで入っているようなので、以下のように書き換えてみる事もできます。本来的には可変長なので、len に応じて byte, word, integer で受けるようにしないとイケませんが、もし len が 3 だったら byte の array を作って対応するしかないのでは、やはり上記の方法が正解なのでしょう。

```
i, k, len: integer;  
j: ^integer;  
  
j := @header_page.var_data[i+2];  
writeln(format(str, [j^]));
```

次に、データベースファイルのリンク先を書き換えている update_db_header() 関数ですが、ここでは new_header_page の fix_data に header_page.fix_data をコピーしてから、clumplets を一つづつコピーして行って、type_byte が 3 の時だけリンク先を変更して書き換えを行います。その後、書き換えがうまくいってれば、データベースファイルの先頭へ page_size 分を書き込んで完了となります。

最後に

今回は、Delphi から Firebird のデータベースファイルを直接操作してみました。本当に触りだけとなりましたが、データベースファイルの内部構造に触れることで、Firebird への理解が少しでも深まって頂けたのなら幸いです。今後は、さらにデータが実際にどのように格納されているのか、インデックスはどのように格納されているのか等、踏み込んだ内容をお伝えできればと考えています。