Space management in InterBase

By Ann Harrison, IBPhoenix

An InterBase® database consists of a set of fixed length pages of different types. Ten page types are currently defined:

- Header page (HDR)
- Data page (DPG)
- Blob page (BLP)
- Transaction Inventory Page (TIP)
- Page Inventory Page (PIP)
- Pointer page (PTR)
- Index Root page (IRT)
- B-tree page (BTR)
- Write-Ahead Log page (LIP))
- Generator page (GEN)

Two of these, page inventory and pointer are used for space management. For those not familiar with InterBase's on-disk structure, the next article, Page Types, includes a brief description of each of the page types.

Page types

All page types include a header that holds generic page information.

```
typedef struct pag {
    SCHAR pag_type;
    SCHAR pag_flags;
    USHORT pag_checksum;
    ULONG pag_generation;
    ULONG pag_seqno; /* WAL seqno of last update */
    ULONG pag_offset; /* WAL offset of last update */
} *PAG;
```

Each specific page type adds more structural information. The first page in every database is its Header page (HDR). Secondary database files also have header pages. Data page (DPG) contain data; Blob page (BLP) contain blob data for those blobs that don't fit on the data page with their parent record. Any data page contains data for only one table. Any blob page contains data for only one blob. Transaction Inventory Page (TIP) contain an array of bits, two per transaction, that indicate the state of the transaction. A transaction id is an index into this array. Every page in the database is represented by one bit in a page inventory page (PIP). The bit indicates whether the page is currently in use. Page inventory pages (PIP) occur at fixed intervals in the database - the interval is determined by the page size. A pointer (PTR) page is the top-level locator for data pages. It contains an array of page numbers for the data pages of the table and a corresponding array of bits that indicate whether the page is full. No pointer page entry is made for blob pages or pages that contain only the second or subsequent pages of data from a fragmented record. Index (IRT) root and B-tree page (BTR) pages are what they appear to be. The only odd thing is that each table can have only one index root page.

For that reason, you can put more indexes on a table when you use a large page size. The log information pages (LIP))) for the Write-Ahead Log page (LIP)) are not currently used, though code to use them is included conditionally. Generator page (GEN) contain arrays of 32 or 64 bit integers, depending on the dialect.

Basic page allocation

Page allocation is handled by the routine PAG_allocate in PAG.C. When some routine needs a new page, it calls PAG_allocate. PAG_allocate gets the page control block from the database block to find the first page information page that has free space. If necessary, it reads that pointer page from disk. It then scans the page, looking for the first free bit, and assigns that page number to the new page. The page image is created in the cache manager (CCH), which give it the appropriate page type. The cache manager then returns the buffer pointer to the routine that requested the new page. When the page is marked for write, the page I/O module (PIO) writes it to the appropriate offset in the database file. *Housekeeping Note:* To keep the database on disk consistent, the pointer page must be written before any page that is allocated from it to avoid doubly allocated pages. Under ordinary circumstances, the shared cache or page locks keep this from happening. If, however, the machine were to crash in mid-operation, the order of page writes can prevent corruption.

Advanced page allocation

If the system does not find space on the first PIP it examines, it reads the next, and so on until it searches the last PIP. If the last unallocated page is the last bit on the last PIP, the routine allocates that page number as the next new PIP, formats it, marks the new PIP as needing to be written and the old PIP as dependent on it. Finally, PAG_allocate calls itself to allocate the page that was requested originally, using the first bit on the new page inventory page. If the database is defined to hold multiple files, when page allocation reaches the end of the first file, it creates a new file, gives it a new header, and resumes allocating pages.

Additional page allocation steps for data pages

A data page is recorded as being in use both in the PIP and in a pointer page for that table. Once the new data page has been marked for write, its page number is written into the first free slot one in the current pointer page or the first free slot on any pointer page. The order of writes is: PIP, data page, pointer-page.

Additional steps for interesting pages

Information about interesting pages is stored in a system table called RDB$PAGES. When an index root page, a transaction inventory page, a Generator page or a pointer page) is created, a new row is stored in RDB$PAGES. This operation can cause a new page, a new pointer page, a new page inventory page or even a new file to be allocated.

Releasing pages

The header page is never released. index root pages, Generator pages and transaction inventory pages are not released either. In theory, they could be, but that would complicate (slightly) some sensitive bookkeeping for (relatively) little gain. Nor are page inventory pages released. Once a database has grown to some size, the only way to shrink it is to recreate it from a backup. When a page is empty, it is put back in the “free space pool” by clearing its bit on the appropriate page inventory page. B-tree pages are released when the index is deleted, deactivated, or rebalanced. Blob pages are released when the blob is released, because the record that owns it is deleted or because the blob itself was modified. Data pages created to hold the trailing part of a fragmented row are released when the row - or at least that version of the row - is removed.

Releasing data pages

When the last row on a normal (non-overflow) data page is deleted, the page is returned to free space in a two-part operation. First, the page is removed from its pointer page, which is the page that associates it with its table. If that empties the pointer page, then the pointer page is also marked as released on its page inventory page. Releasing a pointer page requires changing a system table called RDB$PAGES. RDB$PAGES contains one row for each “interesting” page in the database. Pointer pages, index root pages, generator pages, and transaction inventory pages are considered “interesting”. Releasing an index root page also requires deleting a row from RDB$PAGES. This process can recurse, just as the allocation process recurses, except that neither files nor page inventory pages are released.

Elementary allocation on page

For most of the page types, allocation of space on page is not difficult. Generator pages, transaction inventory pages, page inventory pages, and pointer pages are just [[Array | arrays. When one page fills, another one is allocated. (Theoretic rather than actual in the case of generator pages, but the principle holds). Routines in the module PAG.C manage header pages - they are essentially simple structures followed by a byte array that holds the filenames for secondary files. Space on generator pages and transaction inventory pages is never reused, so there is no reason to look for space on any page of those types except the last. Space on page inventory pages is reused. When a page is released - no longer needed for whatever purpose it had - its entry is cleared. For that reason, the page number of the lowest PIP with space is carried in the database control block. That number is not considered reliable, but a good starting point.

Finding space for data

Each table carries with it a vector of its pointer page numbers, and two high-water marks, one for the first pointer page with data space, and one for the first pointer page with space for a new data page. When storing a record that compresses to less than the page size, DPM looks first for a pointer page with data pages that have free space, then at the header of the pointer page to find the first slot pointing to a page with space.

Now, just a bit more about data pages. Every data page has a header like this:

```
typedef struct dpg {
   struct pag dpg_header;
   SLONG dpg_sequence; /* Sequence number in relation */
   USHORT dpg_relation; /* Relation id */
   USHORT dpg_count; /* Number of record segments on page */
   struct dpg_repeat
   {
   USHORT dpg_offset; /* Offset of record fragment */
   USHORT dpg_length; /* Length of record fragment */
   }  dpg_rpt [1];
} *DPG;
```

The repeating offset/length is an array of pointers to data on the page. These pointers are called line index entries, at least by me. The actual data starts at the bottom of the page and works up. When there is no longer enough space for another line index entry and another minimal sized record, plus whatever space is reserved for future expansion (that's another topic), the page is marked full, both in its header and on the pointer page.

DPM goes through the line index, adding up the space on page. If there's enough for the compressed record, alignment overhead, and a line index entry, it's got a winner. However, the space may not be contiguous. In that case, DPM shuffles all the data down to the bottom of the page. Obviously, it doesn't compress the line index entries, though it does correct the offset for data that has moved. Next step is to create a new line index entry and shoot the data onto the page. Final step is to see if the page's fullness quotient has changed and make appropriate changes if so.

If there is space on page, but not enough for the current compressed record, DPM marches on through the pointer page, checking plausible candidates, then on through other pointer pages until there are no more allocated data pages.

OK, now it's time to allocate a new data page. First, find a free page in the current PIP, or the next PIPs, or create a new PIP. Next, create the page in a buffer. Now, starting with the first pointer page that has space to hold a new data page pointer, or create a new pointer page for the table. That's it. At least that's all I can explain at the moment.