NAME

btree − btree database access method

SYNOPSIS

#include <sys/types.h>
#include <db.h>

DESCRIPTION

The routine *dbopen* is the library interface to database files. One of the supported file formats is btree files. The general description of the database access methods is in *dbopen*(3), this manual page describes only the btree specific information.

The btree data structure is a sorted, balanced tree structure storing associated key/data pairs.

The btree access method specific data structure provided to *dbopen* is defined in the <db.h> include file as follows:

```
typedef struct {
        u_long flags;
        u_int cachesize;
        index_t psize;
        int lorder;
        int minkeypage;
        int (*compare)(const DBT *key1, const DBT *key2);
        int (*prefix)(const DBT *key1, const DBT *key2);
} BTREEINFO;
```

The elements of this structure are as follows:

flags The flag value is specified by *or*'ing any of the following values:

R_DUP

Permit duplicate keys in the tree, i.e. permit insertion if the key to be inserted already exists in the tree. The default behavior, as described in *dbopen*(3), is to overwrite a matching key when inserting a new key or to fail if the R_NOOVERWRITE flag is specified. The R_DUP flag is overridden by the R_NOOVERWRITE flag, and if the R_NOOVERWRITE flag is specified, attempts to insert duplicate keys into the tree will fail.

If the database contains duplicate keys, the order of retrieval of key/data pairs is undefined if the *get* routine is used, however, *seq* routine calls with the R_CURSOR flag set will always return the logical ‘‘first’’ of any group of duplicate keys.

cachesize

A suggested maximum size (in bytes) of the memory cache. This value is **only** advisory, and the access method will allocate more memory rather than fail. Since every search examines the root page of the tree, caching the most recently used pages substantially improves access time. In addition, physical writes are delayed as long as possible, so a moderate cache can reduce the number of I/O operations significantly. Obviously, using a cache increases (but only increases) the likelihood of corruption or lost data if the system crashes while a tree is being modified. If *cachesize* is 0 (no size is specified) a default cache is used.

psize Page size is the size (in bytes) of the pages used for nodes in the tree. The minimum page size is 512 bytes and the maximum page size is 64K. If *psize* is 0 (no page size is specified) a page size is chosen based on the underlying file system I/O block size.

lorder The byte order for integers in the stored database metadata. The number should represent the order as an integer; for example, big endian order would be the number 4,321. If *lorder* is 0 (no order is specified) the current host order is used.

minkeypage
The minimum number of keys which will be stored on any single page. This value is used to determine which keys will be stored on overflow pages, i.e. if a key or data item is longer than the pagesize divided by the minkeypage value, it will be stored on overflow pages instead of in the page itself. If *minkeypage* is 0 (no minimum number of keys is specified) a value of 2 is used.

compare
Compare is the key comparison function. It must return an integer less than, equal to, or greater than zero if the first key argument is considered to be respectively less than, equal to, or greater than the second key argument. The same comparison function must be used on a given tree every time it is opened. If *compare* is NULL (no comparison function is specified), the keys are compared lexically, with shorter keys considered less than longer keys.

prefix
Prefix is the prefix comparison function. If specified, this routine must return the number of bytes of the second key argument which are necessary to determine that it is greater than the first key argument. If the keys are equal, the key length should be returned. Note, the usefulness of this routine is very data dependent, but, in some data sets can produce significantly reduced tree sizes and search times. If *prefix* is NULL (no prefix function is specified), **and** no comparison function is specified, a default lexical comparison routine is used. If *prefix* is NULL and a comparison routine is specified, no prefix comparison is done.

If the file already exists (and the O_TRUNC flag is not specified), the values specified for the parameters flags, lorder and psize are ignored in favor of the values used when the tree was created.

Forward sequential scans of a tree are from the least key to the greatest.

Space freed up by deleting key/data pairs from the tree is never reclaimed, although it is normally made available for reuse. This means that the btree storage structure is grow-only. The only solutions are to avoid excessive deletions, or to create a fresh tree periodically from a scan of an existing one.

Searches, insertions, and deletions in a btree will all complete in O lg base N where base is the average fill factor. Often, inserting ordered data into btrees results in a low fill factor. This implementation has been modified to make ordered insertion the best case, resulting in a much better than normal page fill factor.

SEE ALSO

dbopen(3), *hash*(3), *mpool*(3), *recno*(3)

The Ubiquitous B-tree, Douglas Comer, ACM Comput. Surv. 11, 2 (June 1979), 121-138.

Prefix B-trees, Bayer and Unterauer, ACM Transactions on Database Systems, Vol. 2, 1 (March 1977), 11-26.

The Art of Computer Programming Vol. 3: Sorting and Searching, D.E. Knuth, 1968, pp 471-480.

BUGS

Only big and little endian byte order is supported.