B+ Tree Library for Delphi

Table of Contents

Interface

BplusTree.Intf

It's recommended to use interface unit BplusTree.Intf to work with database. This unit provides clean interface to implementation class located in BPlusTree unit. BplusTree.Adapter unit is needed only as binding for bpt4d.dll and normally is not used.

Interface was designed to be straightforward and easy to use.

There are two base interfaces: IBPlusTree and IBPlusTreeCursor.

Most of functions return status TBPlusTreeStatus. If function returns BP_OK status it means no problem occurred, otherwise you need to check status for more details.

Keys

All keys (and values) are represented as TBytes type from *System.SysUtils* unit. Some functions come with *Raw* suffix and take key or value as Pointer and Size, then convert to TBytes internally and call normal function. These Raw functions will not be described, as they copy normal function.

All keys must be non-empty (not nil). But associated values can be empty (i. e. key will just have no value).

Keys are stored in lexicographical order. For example:

```
11
11,22
11,33
11,33,44
11,44
22
```

Implementing custom keys and values

Any type of key-values can be implemented on top of TBytes. The mandatory rule is: key in tree must always be in Big-Endian format to keep valid key order. Values can be stored in any format. But when read or write keys or values make sure you converted it to or from

system endianness.

Cursors

Cursors are used for range queries. Cursor can be created from existing key or from key either greater or lower than some key. Cursor can have optional **prefix** which makes iteration only over some range of keys containing same prefix.
Cursor iterates over keys and value can be read on demand.

Limitations

Only single threaded access supported.

Getting started

To start work with database first need to create base interace with following procedure:

```
procedure BPlusTreeCreate(out Result: IBPlusTree)
```

Note: database isn't created or opened at this stage.

Next you can use IBPlusTree methods to open or create database and manage keys and values.

See methods table for details.

Methods table

IBPlusTree		
CreateNew	Create new database (overwrite existing one)	
	aFileName	Database file name
	aMaxKeySize	Maximal key size
	aPageSize	Size of storage page, usually 8192
	aPagesInCache	Maximal number of cached pages
	aReadOnly	Is database read-only
OpenExisting	Open existing database	
	aFileName	Database file name
	aPagesInCache	Maximal number of cached pages. Page size is known from the tree.
	aReadOnly	Is database read-only
OpenAlways	If database exists it's same as OpenExisting. If it doesn't CreateNew called. All parameters have same meaning.	
ContainsKey	Check if key is present in the tree. Internally it makes search for exact key match.	
	aKey	Key to search
	Result is True if key found or False if not.	
Put	Put key and associated value. If such key already exists, value is replaced. Function has Raw variant.	
	aKey	Search key
	aValue	Associated value
Delete	Delete value by key. Function has Raw variant.	
	aKey	Search key
Flush	Save changes to storage.	
Cleanup	Cleanup garbage, see Garbage Cleanup	
GetValueSize	Find key and get size of associated value. If key found, result is BP_OK	
	aKey	Search key

	out aSize	Size of value
Get	Get value by key. Result is BP_OK if key found. Function has Raw variant.	
	aKey	Search key
	out aValue	Found value, nil if not found
GetExtremeKey	Get first or last key in tree. Result is True if key found.	
	aFirst	Boolean, True if you want to get first key or False if you want last key.
	out aKey	Found key
FirstKey	Get first key in tree. Result is true if key found.	
	out aKey	Found key
LastKey	Get last key in tree. Result is true if key found.	
	out aKey	Found key
CursorCreateEx	Extended version of cursor creation from key. Result is IBPlusTreeCursor. If no key found it is nil.	
	aKey	Search key cursor will point to
	aKeyPos	TKeyPositions, define how search should happen. Options are: • [kpEqual] • [kpLess, kpEqual] • [kpEqual, kpGreater]
	kpEqual	Cursor created from key equal to aKey
	kpLess	Cursor created from first key lesser than aKey
	kpGreater	Cursor created form first key greater than aKey
	aPrefix	Boolean, if True, aKey is used as range prefix. It means all keys in cursor must start with aKey. If this condition fails, cursor is not created.
CursorCreateFirst	Create cursor from first key in tree.	

CursorCreateLast	Create cursor from last key in tree.
IBPlusTreeCursor	
Next	Return True if next key found and cursor moved to this key.
Prev	Return True if previous key found and cursor moved to this key.
NextPage	Return True if first key at next page found and cursor moved to this key.
PrevPage	Return True if first key at previous page found and cursor moved to this key.
GetKey	Get current key, use Key property.
GetValue	Get current value, use Value property.

Technical details

Database tree consist of two layers: paged storage layer and tree layer itself.

Storage

Paged storage layer deals with page input/output and caching. Page size can be chosen by user for new database or will be read from existing database. Page size must be a multiplier of disk page size for better performance. Page must be big enough to have minimal number of keys.

Minimal page size is 512 bytes, but in real applications it's usually 8192, 4096 or 16384 bytes.

Storage and caching implemented in PagedStorage unit. In general storage is based on stream which can be file or memory or custom stream. The library uses disk based storage.

Cache

Number of pages in cache can be set by user. There is no common formula for optimal number of pages. When there is no space for new page less used page is flushed. Page count can be considered from available RAM. For example 128 pages x 8192 bytes each will occupy 1 MB of RAM. Small databases can fully reside in RAM.

Pages

All storage pages are numbered in ascending order by storage position. The first page is service page which contains database header. User doesn't need direct access to pages or page data. Other pages are either *index pages* or *data pages*. Both type of pages are stored in single storage object (file, for disk based storage).

Index Pages can be *Leaf* or *Branch* pages. Leaf pages contain keys and pointers to associated values, which stored in Data Pages. Branch pages contain keys and pointers to find leaf page.

Key size cannot be larger than page size.
Keys without values (or rather with null values) allowed.

Leaf pages are doubly linked. It allows to easily iterate over the pages.

Search

Search is cheap operation. When tree is built finding a key is fast and simple. Each key is searched starting from root page and guided down by branches to find leaf page which has pointers to values.

Insertion

Insertion is more complicated and expensive operation than search in cases when page splits occurs (if page is full).

Deletion

Deletion is expensive operation in cases when page merging occurs.

On deletion, only key is deleted, value remains at its place but is not indexed anymore. This leads to garbage data left. It does not affect search speed.

Garbage

Due to value deletion or reallocating there can be garbage data in database.

Garbage cleanup

To clean accumulated garbage, database can be cleaned periodically. Cleanup does rewrite all database keys and values into temporary database and then replace original database with temporary.

If database will be read-only, it's enough to clean database once it was filled with data.