pjjTextBase overview

pjjTextBase is a set of powerful yet easy to use 20+ functions that provides a complete management for a **simple relational database** system, with data kept in industry standard **text files** (CSV, flat files). With pjjTextBase you can quickly develop a robust database-driven web application: anything from a guest-book, forum, poll etc. to fully-fledged CMS or e-commerce project. pjjTextBase is **fast, stable, flexible,** Unicode compatible, and comes with **detailed documentation** (html, pdf and Microsoft Help). Perfect solution if you can't or don't want to use SQL–the **only requirement is PHP 4+**, making your application very compatible with a variety of hosting plans and instantly deployable by a simple upload of the scripts.

To store data, **pjjTextBase** uses text files ("flat files"), eg.:

```
id|author|title
1|Agatha Christie|Evil Under The Sun
2|Arthur Conan Doyle|A Study in Scarlet
3|Raymond Chandler|Farewell My Lovely
```

Read more about **data file format** used by **pjjTextBase**.

Why not use SQL?

First of all, to use any SQL solution (MySQL, PostgreSQL etc.) you must have access to one. Although this software is free, you may have to pay extra money to your ISP to use it on your website.

With PHP5 comes another possible solution: SQLite. Unfortunately, not many ISPs offer PHP5-rich accounts.

Sometimes you want something lightweight that will make your project fast. Would you use MySQL only to set a simple counter? Probably not. In such a case **pjjTextBase** could be your best choice.

Databases in simple text-files are easier to maintain: you may use any spreadsheet or Sam Francke's excellent CSV files editor **CSVed** (it can handle also secured CSV files!), which can be downloaded at http://csved.sjfrancke.nl/ Since flat files are text files, you can even edit them with Notepad!

Lastly, **pjjTextBase** is easier to learn, since it has just a few functions, so there's no necessity to learn SQL *yet*.

What can I do with pjjTextBase?

With the script you can manage a normalized database system! You can:

- select records from the database on any condition, even very complex
- count records that follow any given condition, even very complex
- sort table as you like (multiple columns, ASC/DESC)
- add, delete, modify records
- create new tables
- append records from other file
- append fields from other file
- find the maximum and minimum values in given field
- find all unique field values
- add and delete fields
- change field names

Normalized, you say?

Yes! You can not only put your data into one big table (which is trivial): you can actually *normalize* your database, designing a set of linked tables to avoid data redundancy – all these tables will be read into *one* array with just *one* line of code.

Let's suppose you have a file `books.csv`:

```
id|@author|title|%translation
1|1|Evil Under The Sun|1,2
2|1|A Pocket Full of Rye|1,2,3
3|2|A Study in Scarlet|2,3
```

and then two another, called `author.csv`:

```
id|author
1|Agatha Christie
2|Arthur Conan Doyle
3|Raymond Chandler
```

and `translation.csv`:

```
id|translation
1|French
2|Spanish
3|German
```

(I'm far from saying that eg. *Evil Under The Sun* wasn't translated into German; it's just an example, you know.)

All three files should have the same extension.–Mind the @ sign in the header of `books.csv`, as well as field and file name correspondence: `@author`, `author.csv` and `author`! Exactly the same applies to `%translation` field.

When `books.csv` is being connected to, linked file (`author.csv`) is being read as well, so eg. first record will have following fields:

```
[id] -> 1
[@author] -> 1
[title] -> Evil Under The Sun
[author] -> Agatha Christie
[%translation] -> 1,2
```

Linked table can have own link fields, too. For example you could modify the second table in such a way:

```
id|author|@country
1|Agatha Christie|1
2|Arthur Conan Doyle|1
3|Raymond Chandler|2
```

and add another file, called `country.csv`:

```
id|country
1|Great Britain
2|USA
```

so the first record would look as follows:

```
[id] -> 1
```

```
[@author] -> 1
```

```
[title] -> Evil Under The Sun
```

```
[author] -> Agatha Christie
[@country] -> 1
[country] -> Great Britain
[%translation] -> 1,2
```

Now, how about `%translation` field? When `books.csv` was being connected to, `translation.csv` was read as well; I just didn't mentioned this above to explain iteration of connecting via link fields more clearly.

After connecting to all linked files, among them to `translation.csv`, first record looks this way:

```
[id] -> 1
[@author] -> 1
[title] -> Evil Under The Sun
[author] -> Agatha Christie
[@country] -> 1
[country] -> Great Britain
[%translation] -> 1,2
[translation] -> Array:
  [0] -> French
  [1] -> Spanish
```

Let me remind that all this has been done with only one line of code:

```
$books = ptb_connect('books.csv');
```

Now, isn't it great?–Naturally, if you have databases that are not connected via link or multi field, you need to connect to each of them separately.

Unicode support

Actually *Unicode support* means that **pjjTextBase** can read and write data files with BOM = Byte Order Mark, three characters long kind of Unicode signature that may be present at the very beginning of a file. Assumption is being made that *all* or *none* of your data files contain BOM: you can set this up in **ptb_ini.php**.

Regardless your setup, **pjjTextBase** can *read* files with BOM. Notice that Unicode (UTF-8) files in fact *do not need* BOM.

How to set up pjjTextBase?

To use **pjjTextBase** you need just one line:

```
require '/path/to/pjjtextbase.php';
```

where `/path/to/` is, of course, path to a directory, where `pjjtextbase.php` file was placed.

ptb_ini.php

pjjTextBase comes with some default values, which can be changed in `ptb_ini.php` file (I believe it is more flexible to change those values outside the script file: when new version of **pjjTextBase** comes out, you will only have to upload it, whithout any modifications, while your individual settings will remain untouched).

Since version 1.2 they are read sequentially, one by one, which means that you don't have to put all of them into `ptb_ini.php` file, but only these which are different than default ones.

This settings are as follows:

PTB_DEFAULT_DB_LOCATION
: You can use **ptb_connect** without explicitly giving variable **$location**; in such case the script will use this setting. Possible values are: G (directory defined by PTB_PATH_DB), L (local directory, ie. the one where the script is placed) or F (take **$filename** as it is, without $_SERVER['DOCUMENT_ROOT'] dependency, eg. **"../authors.csv"**). Default value is **G**.

PTB_PATH_DB
: Path to a directory, where data files are kept; of course it makes sense only if PTB_DEFAULT_DB_LOCATION (see above) is set to G, in other cases it's just skipped. Default value is **/db**.

PTB_FILE_DB_PREFIX
: What prefix should database files use to stand out more easily? This is also important if you use **link** or **multi fields**. Default value is null (empty string), since you are advised to put all your data files into one directory (described by PTB_PATH_DB)

PTB_NEWLINE
: What kind of newline should PTB use? Windows (\r\n), Unix (\n) or Mac (\r)? Default value is **\n**

PTB_DEFAULT_SECURITY
: Should data files have **security string**? Can be 0 or 1; default is **1**.

PTB_SEC_STR
How the first line for secured files should look like? Default value is **<?php die('Access denied!');?>**

PTB_SHOW_ERRORS
Should the PTB error messages be displayed? Default value is **false** (during developement certainly it should be set to true).

PTB_BOM
Should all written files (ptb_update, ptb_delete *etc*.) have BOM (kind of UTF signature; codepage of data depends on the codepage of your webpage)? Default value is **false**, since UTF-8 files don't really require BOM.

PIPE
If you'd like to use pipe sign somewhere inside a field, you have to use a substitution code for it to avoid confusion with field separator; here you should define this substitution; default is **&pipe;**. Attention! This substitution is switched off to speed up the script; if you need to use it, you have to uncomment one line in **ptb_connect** and comment another.

Line that must be uncommented, if you want to use pipe sign:

```
$record[$recordNo][$fieldnames[$j]] = str_replace(PIPE, '|', $splitLine[$j]);
```

Line that must be commented, if you want to use pipe sign:

```
$database[$recordNo][$fieldnames[$j]] = $splitLine[$j];
```

`ptb_ini.php` file should be placed in the same directory as `pjjtextbase.php`.

Predefined constants

There is (since version 1.2 on) one predefined constant, declared in the main file:

PTB_VERSION
number of current script's version (floating point number)

Data file format

pjjTextBase uses text files ("flat files"), that are basically identical to CSV files (=Comma Separated Values), *ie*. have the following structure: first line contains field names, separated with separator, while records are placed in consecutive lines, eg.:

```
id|author|title
1|Agatha Christie|Evil Under The Sun
2|Arthur Conan Doyle|A Study in Scarlet
3|Raymond Chandler|Farewell My Lovely
```

Althought generally speaking **pjjTextBase** smoothly reads CSV files, there are some limitations and enhancements you should be aware of:

Limitations

1. Separator used by **pjjTextBase** is pipe: |. As such pipe is better than comma (for obvious reason); it is also better than semi-colon, as one can use entities in field values (so perhaps I should refer to PSV files: Pipe Separated Values :-)
2. Quotation marks, surrounding field's value, are not recognized as such: in other words, they are treated as a part of this value.
3. Each record must be put on its own line, so it mustn't contain linebreaks.
4. Fieldname mustn't be a number.
5. Fieldnames `id` and those starting with **@** or **%** have **special meaning**.

Enhacements

1. Data files may contain **security line**
2. Data files may containt comments: each line starting with # is treated as a comment.
3. Data files may containt empty lines (for increased readability).

Please notice that funtions that write down data (**ptb_update**, **ptb_delete**, **ptb_write**, **ptb_addField**, **ptb_delField** and **ptb_changeFieldname**) strip comments and empty lines due to the fact that they simply write down array from the memory; the only exception is **ptb_add**, which *appends* data to a file.

Special fields

There are three special types of fields used by **pjjTextBase**:

1. **id field**
2. **link field**
3. **multi field**

id field

This field is to be treated as a primary key, so it should contain consecutive natural numbers (at least they mustn't be recurrent).

When records are appended from the second table, its `id`s are being renumbered, starting from the greatest `id` of the first table, increased by 1.

link field

Link field, which name starts with **@**, contains single reference to value included in the child table. Child table's name should be the same as link field's (without the @ sign, though):

```
christie.csv - main table:
id|year|@protagonist|title|%aka
1|1920|1|The Mysterious Affair at Styles|
2|1922|3|The Secret Adversary|
3|1923|1|The Murder on the Links|
(...)

protagonist.csv - child table:
id|protagonist
1|Hercule Poirot
2|Miss Marple
3|Tommy and Tuppence
(...)
```

Child tables are read automagically by **ptb_connect**. BTW they can contain more than just two columns: `id` and **link field values**, eg.:

```
protagonist.csv - child table:
id|protagonist|nationality
1|Hercule Poirot|Belgian
```

```
2|Miss Marple|English
3|Tommy and Tuppence|English
(...)
```

You can certainly go on with the normalization of the database by moving protagonists' nationality to `nationality.csv`, which thus would contain child table of the one included in `protagonist.csv`.

Data files with child tables must have the same extension as data file with parent table.

multi field

Multi field can contain multiple references, separated with commas, to values included in some other table. If multi field's name starts with %, the other table will be read automatically. Child table's name should be the same as multi field's (without the % sign):

```
christie.csv - main table:
id|year|@protagonist|title|%aka
(...)
24|1937|1|Murder in the Mews|7
25|1937|1|Dumb Witness|8
26|1938|1|Appointment with Death|
27|1938|1|Hercule Poirot's Christmas|9,10
28|1940|1|One, Two, Buckle My Shoe|11,12
(...)

aka.csv - child table:
id|aka
(...)
7|Dead Man's Mirror
8|Poirot Loses a Client
9|A Holiday for Murder
10|Murder for Christmas
11|An Overdose of Death
12|The Patriotic Murders
(...)
```

Only two columns are read from child table. Data files with child tables must have the same extension as data file with parent table.–To find if a certain reference is included in a multi field, you can use **isThere** or **isThereExt** functions.

Function list:

ptb_add

Adds one record to the **$database**. The record is appended at the end of database file. Proper format (ie. field order and number) is not verified.

Syntax

```
ptb_add($filename, $location, $fieldValues);
```

Syntax description

$filename
: name of file containing the database

$location
: location of **$filename**; can be:

  G

  globally defined (in `ptb_ini.php`) directory

  L

  local directory (ie. the same where script calling **ptb_connect** is placed)

  <path>

  other, defined directory, eg. `/samples/db` (no trailing slash)

  F

  filename (ie. absolute or relative path without DOCUMENT_ROOT dependency)

$fieldValues
: full line of database file, formatted exactly how it is in the database, ie. with field values in proper order (which is set in the header), joined with a separator or (since version 1.2) array (may be multidimentional, eg. **$var[3]**); order of fields is not checked!

Return

This function returns **true** on success and **null** (=false) on failure.

Sample(s)

```
ptb_add('book.php', 'L', '7|M|Bob|Smith|London');
```

See also:

ptb_append • ptb_merge

ptb_addField

Adds **$newFieldname** as a new field to the **$filename**. If the **$newFieldname** already exists, it is, of course, not added.–You probably want to use this function only in some sort of database managment script.

Syntax

```
ptb_addField($filename, $location, $newFieldname[, $defaultValue = '']);
```

Syntax description

$filename
name of file containing the database

$location
location of **$filename**; can be:

G

globally defined (in `ptb_ini.php`) directory

L

local directory (ie. the same where script calling **ptb_connect** is placed)

<path>

other, defined directory, eg. `/samples/db` (no trailing slash)

F

filename (ie. absolute or relative path without DOCUMENT_ROOT dependency)

$newFieldname
name of the new field

$defaultValue
optional: value for the new field (default is null)

Return

This function returns **true** on success and **null** (=false) on failure.

Sample(s)

Add new field `french` to the **christie.csv** file and set its value to 1 for all the records:

```
ptb_addField('christie.csv', 'G', 'french', '1');
```

See also:

ptb_delField • ptb_changeFieldname

ptb_append

Appends table from **$filename** at the end of connected **$database**. If first field of *both* databases is `id` and no **$keyField** is used, then `id` of appended database is renumbered, starting from the biggest value of **$database**'s `id` plus 1. Identity of two databases' structure (ie. field names and number) is not checked. If **$database** is not an array, the other one (being appended) is returned.

Syntax

```
$var = ptb_append($database, $filename[, $location = PTB_DEFAULT_DB_LOCATION[, $recursive = true[, $keyField = '']]]);
```

Syntax description

$database
: variable containing connected database

$filename
: name of file containing the database to be appended

$location
: *optional:* location of **$filename**; can be:

  G

  globally defined (in `ptb_ini.php`) directory

  L

  local directory (ie. the same where script calling **ptb_connect** is placed)

  <path>

  other, defined directory, eg. `/samples/db` (no trailing slash)

  F

  filename (ie. absolute or relative path without DOCUMENT_ROOT dependency)

$recursive
: *optional:* specifies whether linked fields should be read; can be set to **true** (default) or **false**

$keyField
: *optional:* see **ptb_connect** for description; should be used if **$database** was connected to with the same **$keyField**

Return

This function returns **array** on success and **null** (=false) on failure.

See also:

ptb_merge • ptb_add

ptb_changeFieldname

Changes **$oldFieldname** to **$newFieldname**. **$filename** is checked for both names: if it already contains the new name or doesn't contain old one, nothing happens.–You probably want to use this function only in some sort of database managment script.

Prior to version 1.2 this function's name was **ptb_changeFieldName**.

Syntax

```
ptb_changeFieldname($filename, $location, $oldFieldname, $newFieldname);
```

Syntax description

$filename
: name of file containing the database

$location
: location of **$filename**; can be:

  G

  globally defined (in `ptb_ini.php`) directory

  L

  local directory (ie. the same where script calling **ptb_connect** is placed)

  <path>

  other, defined directory, eg. `/samples/db` (no trailing slash)

  F

  filename (ie. absolute or relative path without DOCUMENT_ROOT dependency)

$oldFieldname
: field which name is to be changed

$newFieldname
: new field name

Return

This function returns **true** on success and **null** (=false) on failure.

Sample(s)

Change field aka to other_titles in the **christie.csv** file:

```
ptb_changeFieldname('christie.csv', 'G', 'aka', 'other_titles');
```

See also:

ptb_addField • ptb_delField

ptb_connect

Connects to the specified database, which simply means that data from the database are read into memory as an array. Preliminary condition for displaying records and using some other functions: **ptb_select**, **ptb_map**, **ptb_count**, **ptb_max**, **ptb_min**.

Syntax

```
$var = ptb_connect($filename[, $location = PTB_DEFAULT_DB_LOCATION[, $recursive = true[, $keyField = '']]]);
```

Syntax description

$filename
: name of file containing the database

$location
: *optional:* location of **$filename**; can be:

G

globally defined (in `ptb_ini.php`) directory

L

local directory (ie. the same where script calling **ptb_connect** is placed)

<path>

other, defined directory, eg. `/samples/db` (no trailing slash)

F

filename (ie. absolute or relative path without DOCUMENT_ROOT dependency)

$recursive
: *optional:* specifies whether linked fields should be read; can be set to **true** (default) or **false**

$keyField

optional: name of field to be used as main key; if it is not given, first record's key is 0, next 1 *etc.*; if it is used, each record's key is equal to **$keyField**'s value (usually you'd want to use `id` field here); **$keyField** should be primary key: if there is more than one record with the same value of **$keyField**, only the last one will be inserted into table (**read more**)

Return

This function returns **array** or **null**, if data file doesn't exist or contains no data.

Sample(s)

```
$books = ptb_connect('books.csv');
$books = ptb_connect('books.csv', 'L');
$books = ptb_connect('books.csv', '/other/data');
$books = ptb_connect('../books.csv', 'F');
```

Tips

I find it to be a useful programming custom to name variable that holds the **whole** database after the **$filename**, even if it may be subsequently modified by reading linked files or with **ptb_merge**, eg.

```
$something = ptb_connect('something.csv', 'L');
```

Keyfield concept

If you don't use **$keyField** (4th argument to **ptb_connect**), records' keys start at 0 and are consecutive: 0, 1, 2, 3 etc. The biggest is equal to total number of records in the array minus 1, and you can loop through the array with standard **for**:

```
for ($i = 0, $c = ptb_count($myDatabase); $i < $c; $i++) {
  // instructions, eg. echo $myDatabase[$i]['title'];
}
```

Pure and simple, with one drawback, though: you can't easily get other values of the record, for which one field's value (usually it would be `id`) you know–you need to use **ptb_map** for this. Probably the most common example of this situation is when you have value of $_GET['id'] and want to display other fields of the record, for which `id` is equal to it:

```
$x = ptb_map($myDatabase, "'id' == '$_GET['id']'");
```

Now **$x** holds the said record (**ptb_map** without third argument was introduced in version 1.2), so you can use:

```
echo $x[0]['title'];
```

With setting **$keyField** to `id` you don't need to use **ptb_map** at all, since you can directly acces the record in question, eg.:

```
echo $myDatabase[$_GET['id']]['title'];
```

Of course you can set **$keyField** to *any* field, but it makes sense if it is set to primary key of your databse (in other words to such field, for which values are unique throughout the whole database, so you can doubtlessly identify each record by the said value). If there are eg. two records with identical `id`s, only the latter will be available.

Please notice that if you want to loop through the database with standard **for**, you need to start from the smallest `id`, not from 0:

```
for ($i = <smallest id>, $c = ptb_count($myDatabase); $i < $c; $i++) {
  // instructions, eg. echo $myDatabase[$i]['title'];
}
```

If there are some "gaps", ie. `id`s aren't consecutive, you'll get warning message. In such a case you may want to use **foreach**:

```
foreach ($myDatabase as $key => $value) {
  // instructions, eg. echo $myDatabase[$key]['title'];
}
```

If you use **$keyField** in **ptb_connect**, do not forget to use it also in **ptb_merge**.

ptb_count

Finds number of records in given **$database**. If **$condition** is given, only records that follow it are counted.

Syntax

```
$var = ptb_count($database[, $condition = '']);
```

Syntax description

$database
variable containing connected database

$condition
optional: condition upon which records should be counted

Return

This function returns **integer**; if **$database** is not an array, it returns 0 (unlike **count**, which returns 1).

Sample(s)

```
$c = ptb_count($myDatabase);
$c = ptb_count($myDatabase, "'name' == '{$_GET['name']}'");
```

History

ptb_count accepts **$condition** since **version 1.1**

ptb_create

Creates database file with header only, ie. security string (unless **$secure** is set to 0) and line with field names. If file with **$filename** already exists in the given **$location**, it is deleted. On error of creating the file either writing the header, message is displayed. Notice that in order to have unsecured file, you should set **$secure** to **0**, not to **false**.

Syntax

```
ptb_create($filename, $location, $fieldnamesLine [, $secure =
PTB_DEFAULT_SECURITY]);
```

Syntax description

$filename
name of file containing the database

$location
location of **$filename**; can be:

G

globally defined (in `ptb_ini.php`) directory

L

local directory (ie. the same where script calling **ptb_connect** is placed)

<path>

other, defined directory, eg. `/samples/db` (no trailing slash)

F

filename (ie. absolute or relative path without DOCUMENT_ROOT dependency)

$fieldnamesLine
names of fields, separated by a seperator (ie. pipe |)

$secure
optional: 1 (default value) if file should have security string in its first line; otherwise 0

Return

This function returns **true** on success and **null** (=false) on failure.

Sample(s)

```
ptb_create('country.php', 'L', 'id|country|capital');
```

ptb_delete

Deletes records from the database contained by **$filename** placed in **$location** upon given **$condition**.

Syntax

```
ptb_delete($filename, $location, $condition);
```

Syntax description

$filename
name of file containing the database

$location
location of **$filename**; can be:

G
globally defined (in `ptb_ini.php`) directory

L
local directory (ie. the same where script calling **ptb_connect** is placed)

<path>
other, defined directory, eg. `/samples/db` (no trailing slash)

F
filename (ie. absolute or relative path without DOCUMENT_ROOT dependency)

$condition
condition upon which record(s) should be deleted

Return

This function returns **true** on success and **null** (=false) on failure. Please notice that it returns **true** also when no changes has been made to the **$filename**.

Sample(s)

```
ptb_delete('book.php', 'L', "'author' == 'Agatha Christie'");
```

ptb_delField

Deletes **$fieldname** from the **$filename**.–You probably want to use this function only in some sort of database managment script.

Syntax

```
ptb_delField($filename, $location, $fieldname);
```

Syntax description

$filename
name of file containing the database

$location
location of **$filename**; can be:

G
globally defined (in `ptb_ini.php`) directory

L

local directory (ie. the same where script calling **ptb_connect** is placed)

<path>

other, defined directory, eg. `/samples/db` (no trailing slash)

F

filename (ie. absolute or relative path without DOCUMENT_ROOT dependency)

$fieldname
field to be deleted

Return

This function returns **true** on success and **null** (=false) on failure.

Sample(s)

Delete field french from the **christie.csv** file:

```
ptb_delField('christie.csv', 'G', 'french');
```

See also:

ptb_addField • ptb_changeFieldname

ptb_fieldnames

List names of all fields in the **$database** or tells if a **$fieldname** exists in it.

Syntax

```
$var = ptb_fieldnames($database[, $fieldname = '']);
```

Syntax description

$database
variable containing connected database

$fieldname
optional: name of a field, which existence in **$database** you want to check

Return

This function used with one argument returns **array** or **null** if database is empty; used with two arguments returns **true** is the **$fieldname** exists or **null** (=false) othwerwise.

Sample(s)

```
if (ptb_fieldnames($myDatabase, 'author')) {
  // if field 'author' exists in $myDatabase, proceed
  ...
```

ptb_listUnique

Finds all unique values in given field.

Syntax

```
$var = ptb_listUnique($database, $fieldname);
```

Syntax description

$database
variable containing connected database

$fieldname
name of the field, for which unique values should be found

Return

This function returns **array** or **null** if database is empty.

Sample(s)

```
$authors = ptb_listUnique($myDatabase, 'author');
```

If you need to use this function, it's a sign you should think about changing structure of your database :-)

ptb_map

Returns whole first record for which **$equation** is met. If **$otherFieldname** is given, only value of this field is returned.

Syntax

```
$var = ptb_map($database, $equation[, $otherFieldname]);
```

Syntax description

$database
variable containing connected database

$equation
condition upon which record should be selected, in the form **'fieldname' == 'value'**

$otherFieldname
optional: name of other field, which value you look for

Return

This function returns **array** or, if **$otherFieldname** was given, **string**. If nothing was found, **null** is returned.

Sample(s)

```
$x = ptb_map($myDatabase, "'id' == 5", 'age');
```

The above example finds value of `age` field of the record, for which `id` field equals to 5.

This function is useful, when you know value of `id` (usually as $_GET['id']) and want to find value of other field of this particular record. Notice that you can't just use `id` as the number of the record in the database (precisely speaking, you could try to use id-1 as this number), since `id`s will not always be consecutive natural numbers (could be sorted, deleted *etc*.).

Update: if `id` is your primary key, it's much faster and convenient to use **ptb_connect** with `id` as **$keyField**–this way you don't need to use **ptb_map** at all!

ptb_map is faster then **ptb_select** with **$limit** 1, though to find *all* fields of the record in question it is faster and simpler to use **ptb_select** instead of consecutive **ptb_map**.

ptb_max

Finds the biggest numeric value in given field. (For the smallest value, use **ptb_min**.)

Syntax

```
$var = ptb_max($database, $fieldname);
```

Syntax description

$database
variable containing connected database

$fieldname
name of the field, for which the biggest value should be found

Return

This function returns **integer** when used on field with numeric value or **null** if database is empty.

Sample(s)

```
$lastYear = ptb_max($myDatabase, 'year');
```

ptb_merge

Connects to a **$filename** and "juxtaposes" table from it to the **$database** (ie. adds its fields). First columns of both tables must be equal (usually it will be id s). If **$database** is not an array, the other one (being juxtaposed) is returned. If the other (from **$filename**) isn't, **$database** is returned.

Syntax

```
$var = ptb_merge($database, $filename[, $location = PTB_DEFAULT_DB_LOCATION[, $keyField = '']]);
```

Syntax description

$database
connected database

$filename
name of file containing the second table

$location
optional: location of **$filename**; can be:

G

globally defined (in `ptb_ini.php`) directory

L

local directory (ie. the same where script calling **ptb_connect** is placed)

<path>

other, defined directory, eg. `/samples/db` (no trailing slash)

F

filename (ie. absolute or relative path without DOCUMENT_ROOT dependency)

$keyField
optional: name of field to be used as main key; use if **$database** was connected with **$keyField**, for details see **ptb_connect**

Return

This function returns **array** (unless both aren't arrays, then **null**).

Sample(s)

```
$myDatabase = ptb_merge($myDatabase, 'otherdbase.csv', 'L');
```

See also:

ptb_append • ptb_add

ptb_min

Finds the smallest numeric value in given field. (For the biggest value, use **ptb_max**.)

Syntax

```
$var = ptb_max($database, $fieldname);
```

Syntax description

$database
variable containing connected database

$fieldname
name of the field, for which the smallest value should be found

Return

This function returns **integer** when used on field with numeric value or **null** if database is empty.

Sample(s)

```
$firstYear = ptb_min($myDatabase, 'year');
```

ptb_select

Selects records up to **$limit** from **connected** database according to given condition and optionally sorts them.

Syntax

```
$var = ptb_select($database, $condition[, $sort = ''[, $limit = '']]);
```

Syntax description

$database
: variable containing connected database

$condition
: condition upon which records should be selected

$sort
: *optional:* how records should be sorted (sorting on multiple columns allowed)

$limit
: *optional:* how many records meeting given condition should be picked up (if no **$limit** is given, all such records will be selected)

Return

This function returns **array** or **null**, if nothing has been selected.

Sample(s)

```
  $x = ptb_select($myDatabase, "'age' > 40", 'birth DESC');
  $x = ptb_select($myDatabase, "'age' > 40 AND 'sex' == 'M'", 'name ASC, birth
DESC');
  $x = ptb_select($myDatabase, "even('age')");
  $x = ptb_select($myDatabase, "'name' == '{$_GET['name']}'");
  $x = ptb_select($db_test, "'id' == '{$db_test1[1]['price']}'");
```

Tips

Order of **$limit** and **$sort** arguments doesn't really matter; if you want to set only **$limit**, you can simply use:

```
  $x = ptb_select($myDatabase, "even('age')", 10);
```

If value of multidimensional table is to appear in **$condition**, use {} to surround the expression (see the last example above; braces were rendered bold to increase visibility).

If you need only **one** record, always use **$limit** = 1 to speed up the function.

If you need **all** the records, you needn't use **ptb_select** at all, since the variable you used to connect to the data file already contains them!

If you want to find value of one (unique or just first) field of a record, in which another field is equal to a given value, you should use **ptb_map** instead.

If you want to search the database for records containing specified text in certain field, eg. `author`, you can use eregi() function:

```
$x = ptb_select($myDatabase, "eregi('text_to_look_for', 'author')");
```

ptb_sort

Sorts **$database** on multiple columns.

Syntax

```
$var = ptb_sort($database, $sort);
```

Syntax description

$database
variable containing connected database

$sort
how records should be sorted: set of pairs: `fieldname` and ASC/DESC, separated with commas (see sample)

Return

This function returns **array**.

Sample(s)

```
$myDatabase = ptb_sort($myDatabase, 'name ASC, birth DESC');
```

Limitation

You can't sort on link (@ fieldname) nor multi (% fieldname) field (but you can on linked field).

Tips

If no last sorting sequence (i.e. ASC or DESC) is given, ASC is used:

```
$myDatabase = ptb_sort($myDatabase, 'name ASC, birth');
```

is equal to:

```
$myDatabase = ptb_sort($myDatabase, 'name ASC, birth ASC');
```

If you select any records from your database, you can do sorting in one go with **ptb_select**. In fact, **ptb_select** simply uses **ptb_sort** internally.

ptb_update

Updates records which meet given **$condition**.

Syntax

```
ptb_update($filename, $location, $condition, $update);
```

Syntax description

$filename
: name of file containing the database

$location
: location of **$filename**; can be:

G

globally defined (in `ptb_ini.php`) directory

L

local directory (ie. the same where script calling **ptb_connect** is placed)

<path>

other, defined directory, eg. `/samples/db` (no trailing slash)

F

filename (ie. absolute or relative path without DOCUMENT_ROOT dependency)

$condition
condition upon which records should be selected

$update
new values for updated fields given in the form **'field' = 'value'**; consecutive updates should be separated with commas

Return

This function returns **true** on success and **null** (=false) on failure. Please notice that it returns **true** also when no changes has been made to the **$filename**.

Sample(s)

```
  ptb_update('author.php', 'L', "'author' == 'Arthur Conan Doyle'", "'country' =
'3'");
```

You can change values of more than one field at a time; new assignments should be separated with commas:

```
  ptb_update('books.php', '', "'id' == {$_GET['id']}", "'stock' =
{$_POST['changeStock']}, 'price' = {$_POST['changePrice']}");
```

Limitations

You can't use functions as a replacement directly; this won't work:

```
  ptb_update('books.php', '', "'id' == '5'", "'published' = date('Y')");
```

To insert function's result, you must use new variable:

```
  $year = date('Y');
  ptb_update('books.php', '', "'id' == '5'", "'published' = $year");
```

The variable value is treated as a string, i.e. you can't use this:

```
  $year = date('Y');
  ptb_update('books.php', '', "'id' == '5'", "'published' = ($year + 1)");
```

You can still use functions in **$condition**, though:

```
$year = date('Y');
ptb_update('books.php', '', "'id' == date('Y') + 1", "'published' = $year");
```

ptb_write

Writes database to a file. Useful, when you made some operations on a database (like selecting, sorting *etc*.) and want to save the result.

Syntax

```
ptb_write($filename, $location, $table[, $security = PTB_DEFAULT_SECURITY]);
```

Syntax description

$filename
: name of file containing the database

$location
: location of **$filename**; can be:

  G

  globally defined (in `ptb_ini.php`) directory

  L

  local directory (ie. the same where script calling **ptb_connect** is placed)

  <path>

  other, defined directory, eg. `/samples/db` (no trailing slash)

  F

  filename (ie. absolute or relative path without DOCUMENT_ROOT dependency)

$table
: variable containing the database (array)

$security
: *optional:* 1 (default value) if file should have security string in its first line; otherwise 0

Return

This function returns **true** on success and **null** (=false) on failure.

isThere

Finds if a **multi field** contains a certain reference or value. Should be used as a function in condition of **ptb_select**.

Advanced version of this function is **isThereExt**.

Syntax

```
$var = ptb_select($database, "isThere('<needle>', 'fieldname')");
```

Syntax description

$database
variable containing connected database

<needle>
reference or value looked for

fieldname
actual name of multi field

Sample(s)

Find all records belonging to the category, which code is 5 (they may belong to more categories than one, though, since it's a multi field!):

```
$var = ptb_select($myDatabase, "isThere('5', '%category')");
```

Find all records belonging to the category "sci-fi" (again, they may belong to more categories than this one):

```
$var = ptb_select($myDatabase, "isThere('sci-fi', 'category')");
```

isThereExt

Finds if a **multi field** contains a reference or value, meeting the **$condition**. Should be used as a function in condition of **ptb_select**. Simple version of this function is **isThere**.

Syntax

```
$var = ptb_select($database, "isThereExt('fieldname', $condition)");
```

Syntax description

$database
variable containing connected database

$condition
condition upon which records should be selected

In **$condition** instead of sought value of a **fieldname** you should always use **\$piece** (see samples below)!

Sample(s)

Find all records belonging to the category, which code is greater than 5 (they may belong to more categories than one, though, since it's a multi field!):

```
  $var = ptb_select($myDatabase, "isThereExt('%category', '\$piece > 5')");
```

Find all records, for which name of **category** contains 'fiction':

```
  $var = ptb_select($myDatabase, "isThereExt('category', 'eregi(\'fiction\',
\$piece)')");
```

Error messages

Since version 1.2 on error messages are divided into three groups:

1. syntax errors
2. file errors (read/write or filesystem)
3. database structure errors

Please notice that these error messages are not meant to substitute PHP own error messages, but rather come as a useful complement. Be sure to set your PHP errolevel to E_ALL during development (**read in PHP manual how to do it**).

Syntax errors

Sample error message:

```
Syntax error called from
   ptb_create
in file
   <path to file>/index.php
on line no.
   132: ptb_create('novels.csv', 'L');
message:
   this function needs at least 3 arguments!
```

These errors occur when you improperly call a function, eg. give too little arguments, use forbidden char or omit required one, eg.

```
ptb_create('novels.csv', 'L');
ptb_update('novels.csv', 'L', "'id' > 10", "'year' == '1970'")
```

These samples correctly should look like these:

```
ptb_create('novels.csv', 'L', 'id|title|%aka|@protagonist|year');
ptb_update('books.csv', 'L', "'id' > 10", "'year' = '1970'")
```

These errors are always fatal, which means that **pjjTextBase** will always die() on them, no matter if **PTB_SHOW_ERRORS** is set to **true** or **false**.

How to avoid: consult your documentation! ;-)

File errors (read/write or filesystem)

Sample error message:

```
File error (read/write or filesystem) called from
   ptb_connect
in file
   <path to file>/index.php
on line no.
   132: $novels = ptb_connect('novelz.csv', 'L');
message:
   file <path to file>/novelz.csv does not exist!
```

These errors occur when **pjjTextBase** can't find a file, can't open a file or can't write to a file.

These errors do not stop your script from executing. You are thus advised to use conditional structures to catch the error and use appropriate means, eg.

```
if (!ptb_add('novels.csv', 'L', '11|Hercule Poirot\'s Christmas|3,4|1|1938');)) {
  echo 'Server error, can\'t add record to the database.';
  echo 'Please contact webmaster at webmaster@mydomain.net';
}
```

How to avoid: in most cases file errors (especially when a file can't be read or opened) are due to too restrictive file or directory permissions–make sure that your script is allowed to open or write to a file and use chmod if necessary.

Database structure errors

Sample error message:

```
Database structure error called from
   ptb_map
in file
   <path to file>/index.php
on line no.
   132: $test = ptb_map($novels.csv, "'id' == 3", 'country');
message:
   field country does not exist in this database
```

These errors occur when you make reference to a field in your database that does not exist, eg.

```
$database = ptb_connect('novels.csv', 'L', false, 'country');
```

or when you want to change field's name to the one which *does* exist:

```
ptb_changeFieldname('novels.csv', 'L', 'id', 'year');
```

Another (and last) case is when you use variable supposed to be an array but, in fact, isn't.

These errors do not stop your script from executing.

How to avoid: in most cases database structure errors are due to programmer's mistake (eg. misspelled field name).

This version (1.2) comes with no error stack nor error logging; *perhaps* they will be available in future releases.

Security

To assure privacy of your data, you may take three different approaches:

Use .htaccess to prevent access to a data directory

This approach consist in making a directory, eg. **db**, and putting all your data files there, along with a file called .htaccess (yes, its name starts with a dot), with these two lines inside:

```
order deny,allow
deny from all
```

That's it, no one can see them now (but your scripts can!), even if he guesses their name (and the name of the directory).

Place data outside <public_html> directory

Since browser cannot display files that are located outside the <public_html> directory, you may place your data files there and that's it again. Obviously, your scripts will still be able to read and write these files.

If your directory outside <public_html> directory is named eg. **db**, you need to add **/..** to be able to reach it; probably the best idea would be to do it in **ptb_ini.php** file:

```
define('PTB_PATH_DB', '/../db');
```

Use *security string* in your database's header

If no one knows names of your files, he doesn't know how to display them in the browser, right? (names of your files obviously are written in your PHP scripts, but what the user sees are only xHTML pages–because you *do* use xHTML, don't you?–with no variable names). Let's suppose though, that (a) the malicious user guessed the name and location of your files, or (b) that you were lazy enough not to prevent directory listing, or (c) there was an error and the error message was displayed, showing names of your database files–what now? well, really nothing :-) provided that you followed two simple security rules:

1. always use .php extension for your database files (or make your server process them as if they were .php)
2. always use security string

What is this *security string*, anyway? It's simply one line:

```
<?php die('Access denied!');?>
```

placed in the beginning of your database file. With .php extension your files will be parsed by PHP, which means that the only thing an attacker would see is **Access denied!** message.

So to sum it up–yes, I believe you can treat your data as relatively safe. Of course you should consider additional security measures, eg. do not keep passwords in plaintext in your files (but that's another story).

Tweaking script

Some features may be built in **pjjTextBase**, but come switched off or *vice-versa*: sometimes you might want to switch them on or off for yourself–meaning manually edit the script and uncomment/comment appropriate lines, eg. with /* */. These features will be described here.

Please bear in mind that **most probably you don't want to change** the default settings, though.

Features that you might want to switch on

Pipe sign encoding

Sometimes you need to write pipe sign | into the data file as a part of a field value. You may not, however, use it in the straightforward way, since it would mess up record structure. What you have to do is to encode the sign with PIPE constant, as defined in `ptb_ini.php`. **pjjTextBase** can revert it back for you during connecting your data file, but this feature is switched off by default for performance reason. To switch it on, you have to uncomment one line of **pjjTextBase** code and comment another: it is described in detail in **How to set up pjjTextBase** chapter.

Bug reporting guidelines

I like getting bug reports (not that I get many ;-) since it is good way to make **pjjTextBase** better–and I want to continuously improve my script. However, before you let me know that something doesn't work as you expected, please make sure:

1. you use the latest version of pTB
2. you have error reporting switched on (PTB_SHOW_ERRORS is set to **true** in `ptb_ini.php`)
3. you have error displaying set to "on" and error reporting level set to "all" in your `php.ini`:

```
4.     display_errors = On
5.     error_reporting = E_ALL
```

   or you have following line in your script (equivalent to above setting, but afects only current script):

```
error_reporting(E_ALL);
```

   You can read about error reporting in the **PHP manual**.

6. you get your (unexpected) results in browser and not through some IDE
7. your data isn't coded with UTF-16

Please let me know:

- what operating system you use (Windows, Unix, Mac)
- what server you are running (Apache, IIS, lighttpd...)
- how your `ptb_ini.php` looks like

Licence

pjjTextBase is licenced under **The GNU General Public License (GPL)** ver. 2 or any later.

You may use it freely for your private and commercial projects.

Disclaimer

While I have made many efforts to deliver a high quality product, I do not guarantee that **pjjTextBase** is free from defects. The software is provided "as is," and you use the software at your own discretion and risk. I make no warranties as to performance, merchantability, fitness for a particular purpose, or any other warranties whether expressed or implied. No oral or written communication from or information provided by me shall create a warranty. I assume no responsibility for errors or omissions in the software or documentation available from this web site. Under no circumstances shall I be liable for direct, indirect, special, incidental, or consequential damages resulting from the use, misuse, or inability to use this software, even if I have been advised of the possibility of such damages.

Having said that, let me stress that I have been using **pjjTextBase** for over three years in production environment and for me it proved to be a good and reliable software.

History

- new
- corrected

1.2 (June 20th, 2007)

- **ptb_connect** can use 4th argument: **$keyField** with fieldname to be treated as main key of the connected table – thanks go to Wayne from **Team Wishbone** for encouraging me to do it at last
- **ptb_map** with two arguments returns whole record
- **ptb_add** now accepts array as third argument (can be multidimentional, eg. **$var[3]**); order of fields is not checked!
- new function: **ptb_fieldnames**
- new constant: PTB_VERSION
- settings from `ptb_ini.php` are now read one by one, which means that you can put there only those that differ from default ones
- error messages are now divided into three parts: syntax (always fatal), file and database structure (which can be switched off)
- functions that write to files (ptb_create, ptb_add, ptb_update, ptb_delete, ptb_write, ptb_addField, ptb_delField and ptb_changeFieldname) now return **true** on success and **null** (=false) on failure, so you can take appropriate action in case write failed:

```
if (!ptb_create('country.php', 'L', 'id|country|capital')) {
  echo 'Cannot create file.';
  // email yourself or whatever
}
```

- **ptb_select**, **ptb_max** and **ptb_min** on empty database returned 0; now it is null
- function **ptb_changeFieldName** was renamed to **ptb_changeFieldname**
- function **ptb_connect** no more uses **realpath**, since it is blocked on some servers, so one is unable to use this function with L switch; besides it didn't seem to give any real advantage; thanks go to **Evert Everts**
- some other small bugs were corrected

1.1 (November 22nd, 2006)

- more informative error messages
- **ptb_count** now accepts second argument: **$condition**
- new option for data file location: F (absolute or relative path without DOCUMENT_ROOT dependency) – thanks go to JoÃ£o Nuno de Oliveira e Silva for the idea
- **additional check** for the proper use of **$condition**

- in **ptb_connect** extension of linked file (thru link or multi field) was being determined improperly, when path contained dot
- if **multi field** was empty, linked field had improper value
- in **ptb_update** one couldn't insert values with equal sign or comma; now **$update** is parsed in a way that makes it possible
- some other small bugs were corrected

1.0 (July 24th, 2006)

Initial release.

Internet site of pjjTextBase is http://www.pjj.pl/pjjtextbase/