

Game Maker Simple Database 1.2.0

A database system written entirely in GML - By Homunculus

[Introduction](#)

[Installation & How to use](#)

[Reference](#)

» [Generic functions](#)

» [Tables](#)

» [Records](#)

» [Helper functions](#)

[Credits](#)

Introduction

GMsDB is a database system written entirely in GML, thought to be used in any kind of project requiring a small and simple solution to store moderate well-organized amount of data.

GMsDB provides a set of scripts to store, update and retrieve data in an organized way by applying filters like custom conditions, sorting options, limits, offsets, etc.

The system offers functionalities to save the data to file, but the database itself is based on the internal GM data structures, therefore it lives in the system memory and not on the HD. This means that, when the game is closed, the data stored in the database completely disappears unless you explicitly save it to file. This also means however that the data retrieval is not affected by the file management overhead and allows for quite fast reading / writing operations.

Installation & How to use

GEX version: With the GEX you still have to import the .gml file that comes with the extension. This file includes the standard operation scripts that can optionally be modified or extended by the user.

Scripts version: If you are using the scripts directly and not the GEX, you just have to import the .gml file contained in the zip.

Once you have imported the scripts into your game from the provided file, you have to initialize the system by calling `db_init()`. This function creates the required data structures where the database is stored, and must be called only once and before any other script.

Once initialized, you can start creating any number of tables and add records into them with `db_table_create(name,columns)`.

When you don't need the database anymore, you can unload it and free the memory by calling `db_free()`. This deletes all the database tables and values, be sure to save them to file if you don't want to lose the data.

Here's a simple example to get you started:

```
db_init(); //initialize the database. Should be called before any other db call
and only once.

db_table_create("weapons","name|string,weight|real,type|string"); //creates a
table called "weapons" with 3 attributes (columns): name,weight,type

db_record_insert("weapons","excalibur",30,"sword"); //inserts a weapons of type
sword and weight 30 named excalibur
db_record_insert("weapons","lucky sword",20,"sword"); //another sword
db_record_insert("weapons","golden axe",54,"axe"); //an axe
```

```
first_query = db_record_find_ids("weapons",-1,"","weight DESC",-1,0); //returns
a ds_list of weapon ids ordered by weight
```

```
second_query = db_record_find_values("weapons","id,name",db_op_equal,"
type|sword","", -1,0); //returns a ds_grid containing the id and the name of
every sword in the database.
```

```
third_query =
db_record_find_values("weapons","id,name,type,weight",db_op_gt,"weight|25" , "", -
1,0); //returns a ds_grid containing id,name,type and weight of every weapon in
the database with weight greater than 25.
```

```
excalibur_id = db_record_find_first("weapons","name" ,"excalibur"); //returns
the id of the first weapon with name excalibur.
excalibur_weight = db_record_get("weapons",excalibur_id," weight"); //returns
the weight of the sword found in the query above.
```

Reference

Generic functions

db_init()

Initializes the database. Must be called only once, and before any other function.

db_free()

Unloads the database system, deleting all tables and values and freeing its memory.

Tables

A table can be seen as a 2D array (or grid) of values with named columns, kind of like an excel table. In GMSDB tables are identified by their name, a string value, and are available globally. This means that you don't need to store table ids or anything like that when creating tables, in order to manipulate them or manage table data you just need to provide the table name as a string to the scripts and the system will automatically address the right data, independently from the object you are making the call.

db_table_clear(table_name)

Deletes all the records in table, resetting the index but keeping the table structure.

db_table_column_names(table_name)

Returns a ds_list of ordered column names for the specified table. The list can be safely destroyed after usage.

db_table_create(name,columns)

Creates a table in the database having the specified columns. If a table already exists with the same name, it is deleted and replaced.

The special column "id" is auto generated and should NOT be passed in the list of columns.

Columns have to be passed as a string of comma separated name|type values, like in

"firstname|string,lastname|string,age|real, ". A column consists of a name and a type, where the type can

either be the "string" or "real", telling the type of contents to be held in that column.

Example: `db_table_create("fruits","name | string,color | string,weight | real")`

db_table_delete(table_name)

Deletes a table and all its data, freeing the memory.

db_table_exists(table_name)

Returns true if a table with the provided name exists in the database.

db_table_read(filename)

Loads a table and all its contents from a file created with the function `db_table_write()`.

db_table_reindex(table_name)

Used internally. Updates the index of a table.

db_table_size(table_name)

Returns the number of records stored in the table.

db_table_to_csv(table_name,path,sep)

Saves a table content to a CSV file. Values are separated based on the sep argument. Useful for debugging or viewing data in excel or similar programs.

db_table_write(table_name,path)

Saves a table and all its contents to a file. This file can be later used with the function `db_table_read()` to load table into the database.

Records

A record is a row of values stored into a database table. Every record get automatically a unique numeric id when inserted, stored into the id column of the table. Every id is unique, even in the case you delete a record that record id will never be used by other new records.

db_record_delete(table_name,id)

Deletes the record with id from the table.

db_record_exists(table_name,column,value)

Returns true if a value exists in the table at the specified column.

db_record_find_first(table_name,column,value)

Returns the id of the first record having value at the specified column. -1 is returned if no record is found.

db_record_ids(table_name,cond_script,condition,sorting,limit,offset)

Returns a `ds_list` of record ids based on the provided filters / parameters.

table: the table to use for the search.

cond_script: the script used for filtering and comparing the values. -1 means no condition applied.

The following comparison scripts are available:

db_op_equal: checks for equality

db_op_gt: checks for db values greater than a number

db_op_gte: checks for db values greater than or equal to a number

db_op_lt: checks for db values less than a number

db_op_lte: checks for db values less than or equal to a number

db_op_between: checks inclusion of db values between two specified numbers

db_op_not_equal: checks for values different than a provided value

condition: condition to apply for filtering the results, and the column it affects. This is a string in the format "column | condition_value". If for example you want to filter all results having a weight greater than or equal to 100, you have to use `db_op_gte` as `cond_script` and set this argument to "weight | 100".

sorting: string specifying the sorting column and the sort order in the format "column DIRECTION", where direction is either DESC or ASC. If for example you want to sort results by name in ascending order, this becomes "name ASC".

limit: integer telling the maximum number of results returned. A value less than 1 means no limit. **offset**: integer telling the offset to apply to the results (it basically skips the first N results). 0 applies no offset to the results.

Example: `ds_record_find_ids("fruits",db_op_equal,"color | red","weight ASC",-1,0)`

Returns a `ds_list` of all the records in the fruits table having "red" in the "color" column, ordered by weight.

db_record_values(table_name,select,cond_script,condition,sorting,limit,offset)

This is like the above function `db_record_ids`, but returns instead a `ds_grid` of record values instead of `ds_list` of ids. The `select` argument is a string of comma separated columns to return.

Example: `ds_record_find_values("fruits","color,weight",db_op_equal,"name | red","weight ASC",-1,0)`

Returns a `ds_grid` containing the name and the weight of all the red fruits, ordered by weight.

db_record_get(table_name,id,column)

Returns the value in the specified column of record id. -1 is returned if no record with the provided id is found.

db_record_insert(table_name,values)

Inserts a new record in the database table with the provided values (should be passed in column order). Values can either be passed as string of comma separated values, or as script arguments. If passed as string of comma separated values, CSV format applies, this means you can skip special characters like in the CSV specification. It is also possible to sanitize the values by passing them to the script `csv_compose_line()`.

Example (values as arguments): `db_record_insert("fruits","banana","yellow",30)`

Example (values as string): `db_record_insert("fruits","banana,yellow,30")`

db_record_insert_csv(table_name,csv_file,separator)

Populates a table in the database with the contents of a CSV file.

db_record_select(table_name,ids_list,columns)

Given a list of record ids, returns a `ds_grid` containing the values of the provided columns for every id. Columns have to be passed as comma separated names, as in "id,name,color,weight"

db_record_update(table_name,id,columns,values)

Updates the values of the specified columns for record id. Columns have to be passed as a string of comma separated values. Values can either be passed as string of comma separated values, or as script arguments. Returns -1 if no record found.

As in `db_record_insert`, when passed as string, values can be sanitized using `csv_compose_line()`.

Example (values as arguments): `db_record_update("fruits",3,"name,color","banana","yellow")`

Example (values as string): `db_record_update("fruits",3,"name,color","banana,yellow")`

Helper functions

csv_compose_line(list,separator)

Takes a ds_list of values and converts it into a standard CSV string.

csv_parse_line(line,separator)

Returns a ds_list of values from a standard CSV line.

Credits

Developed by Homunculus, no need to credit if used (both free and commercial), but I'd be happy if you send me a note when used in a released project.

Contact me on the [GMC forums](#) or by email at **simoneguerra<at>ekalia.com**