Godot Database Manager

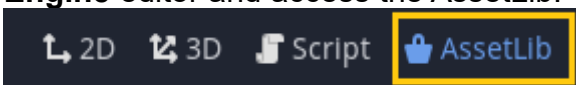
Description:

Godot Database Manager is a plugin for **Godot Game Engine** (https://godotengine.org/) that can create local databases stored in JSON files.

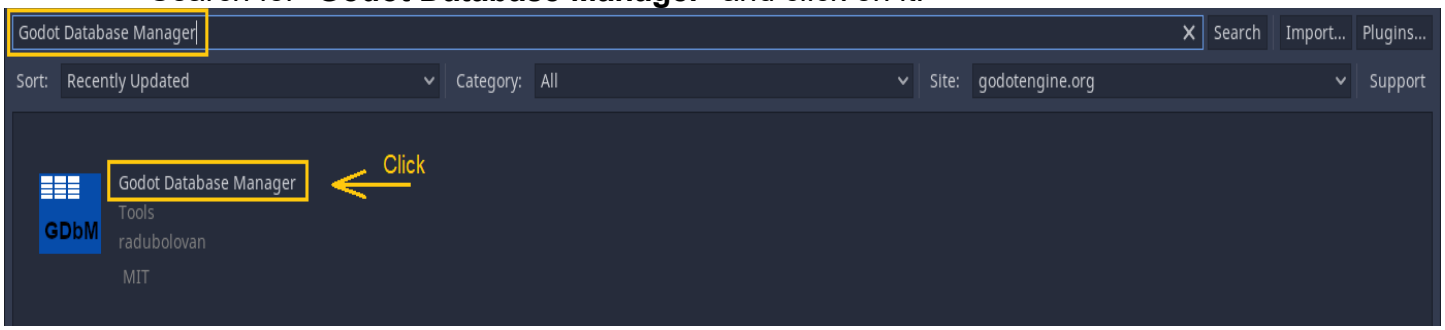
Installation:

There are two ways to download and install the plugin into your project.

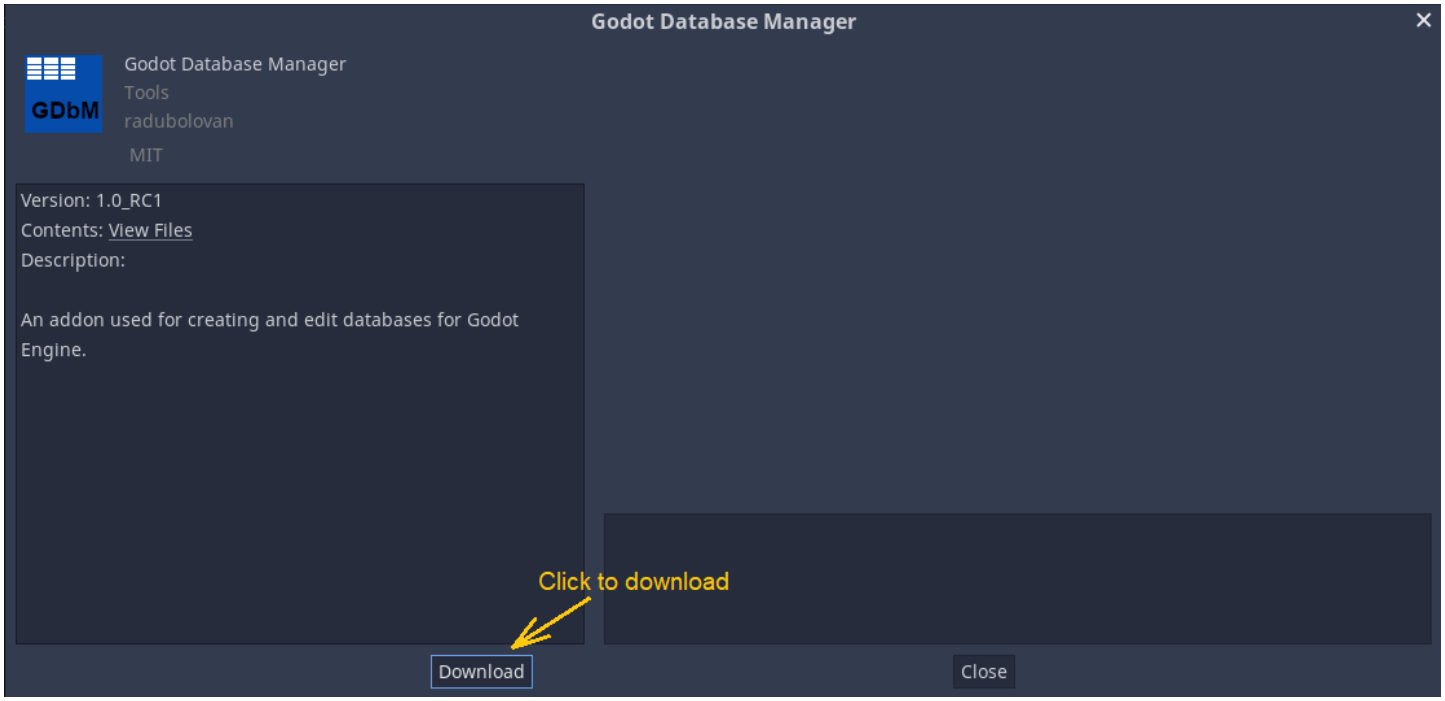
1) Directly in the **Godot Game Engine**'s editor through **Godot Asset Library**:
   - Open **Godot Game Engine** editor and access the AssetLib.



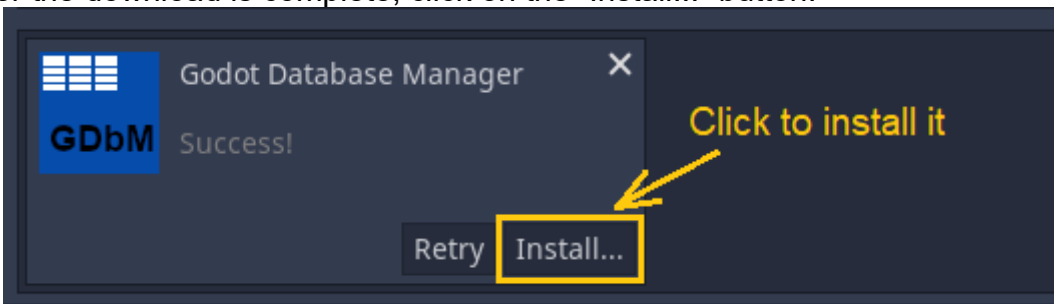
   - Search for "**Godot Database Manager**" and click on it.

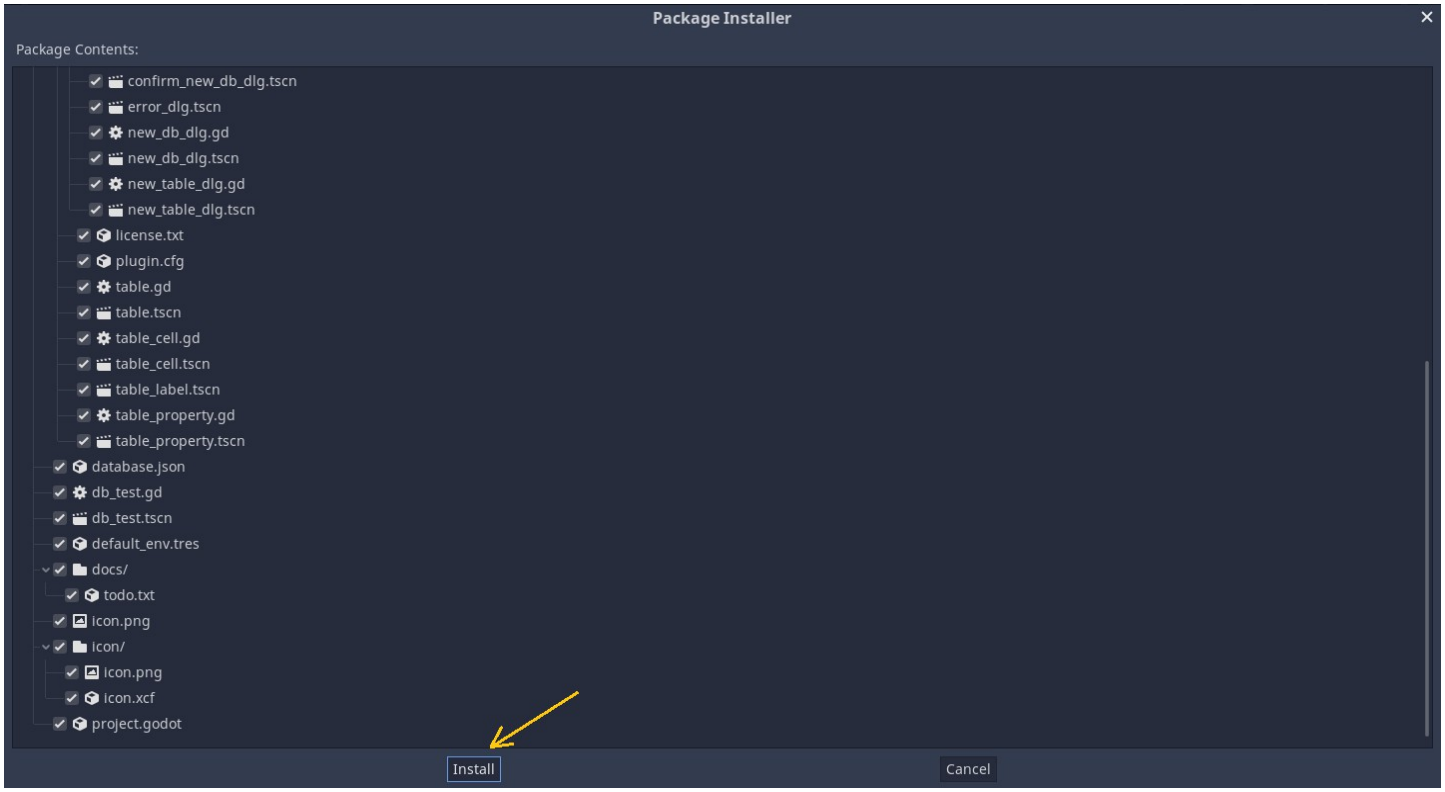


   - Click on the "Download" button.

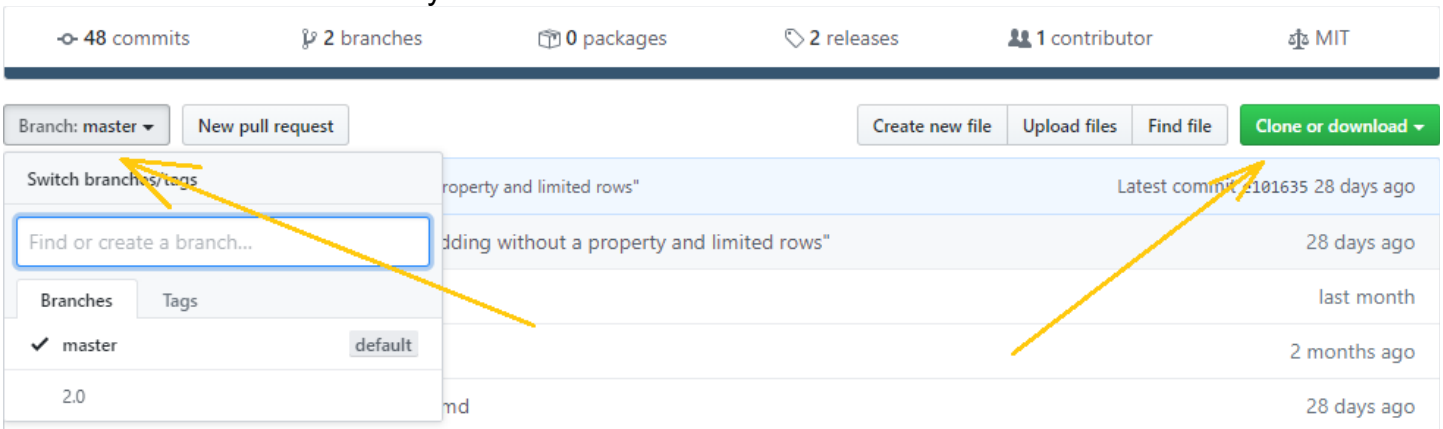


- After the download is complete, click on the “Install...” button.





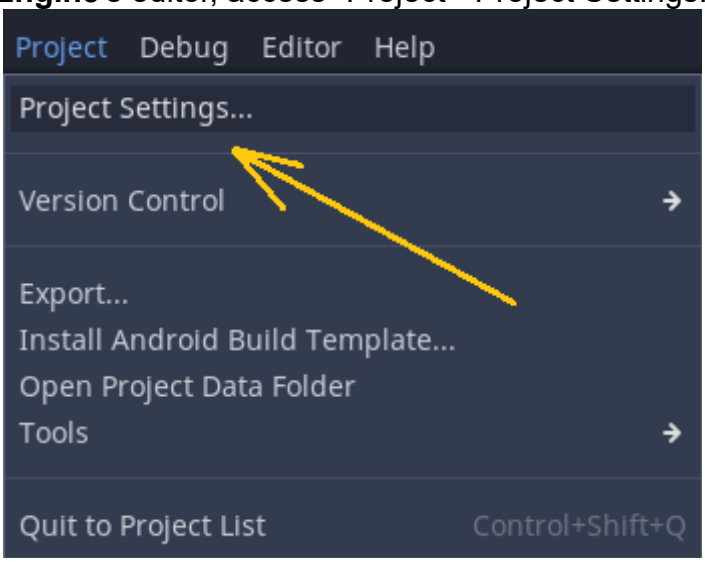
2) Download it from Github
   - You can download it from: https://github.com/radubolovan/Godot-Database-Manager
   - Choose the branch you want and download the archive



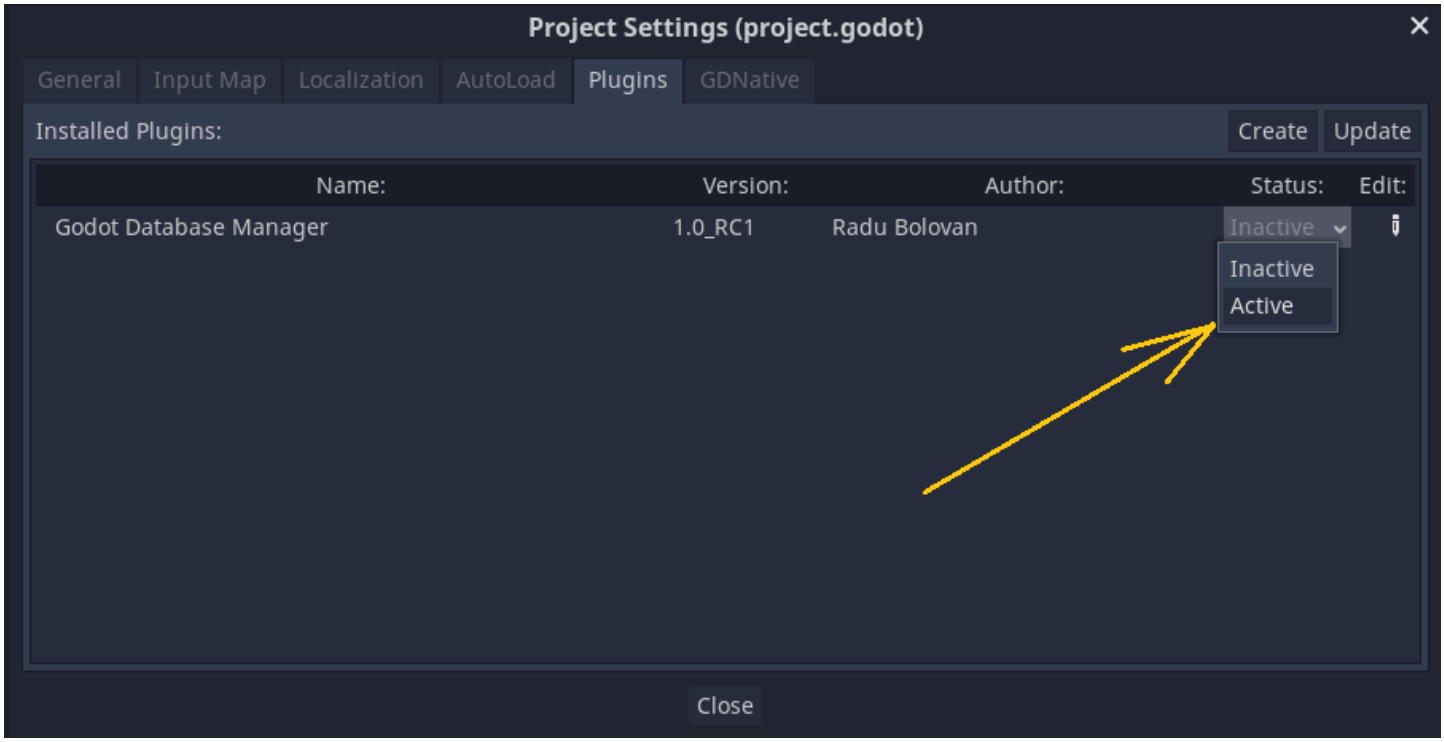
   - Copy the "addons/godot_db_manager" directory (folder) into your project.

Activation:

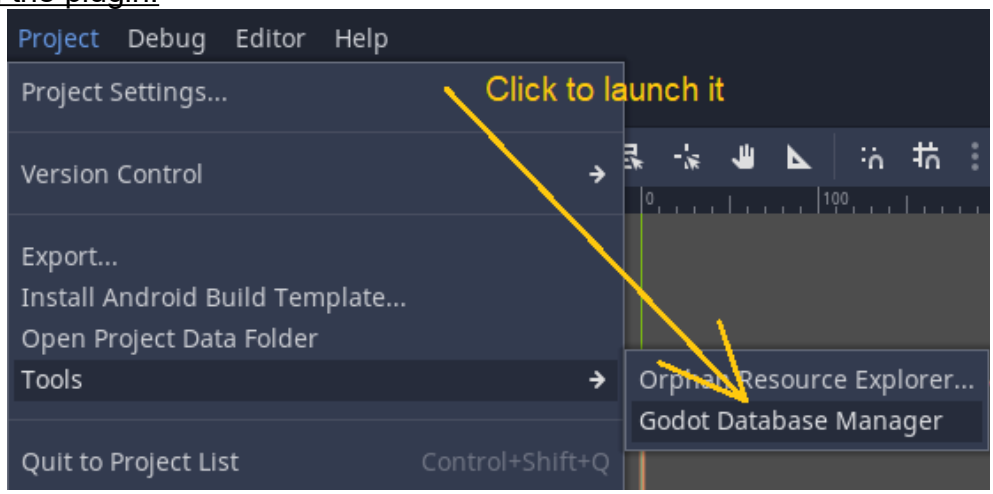
From **Godot Game Engine**'s editor, access “Project->Project Settings...”



… and activate it.

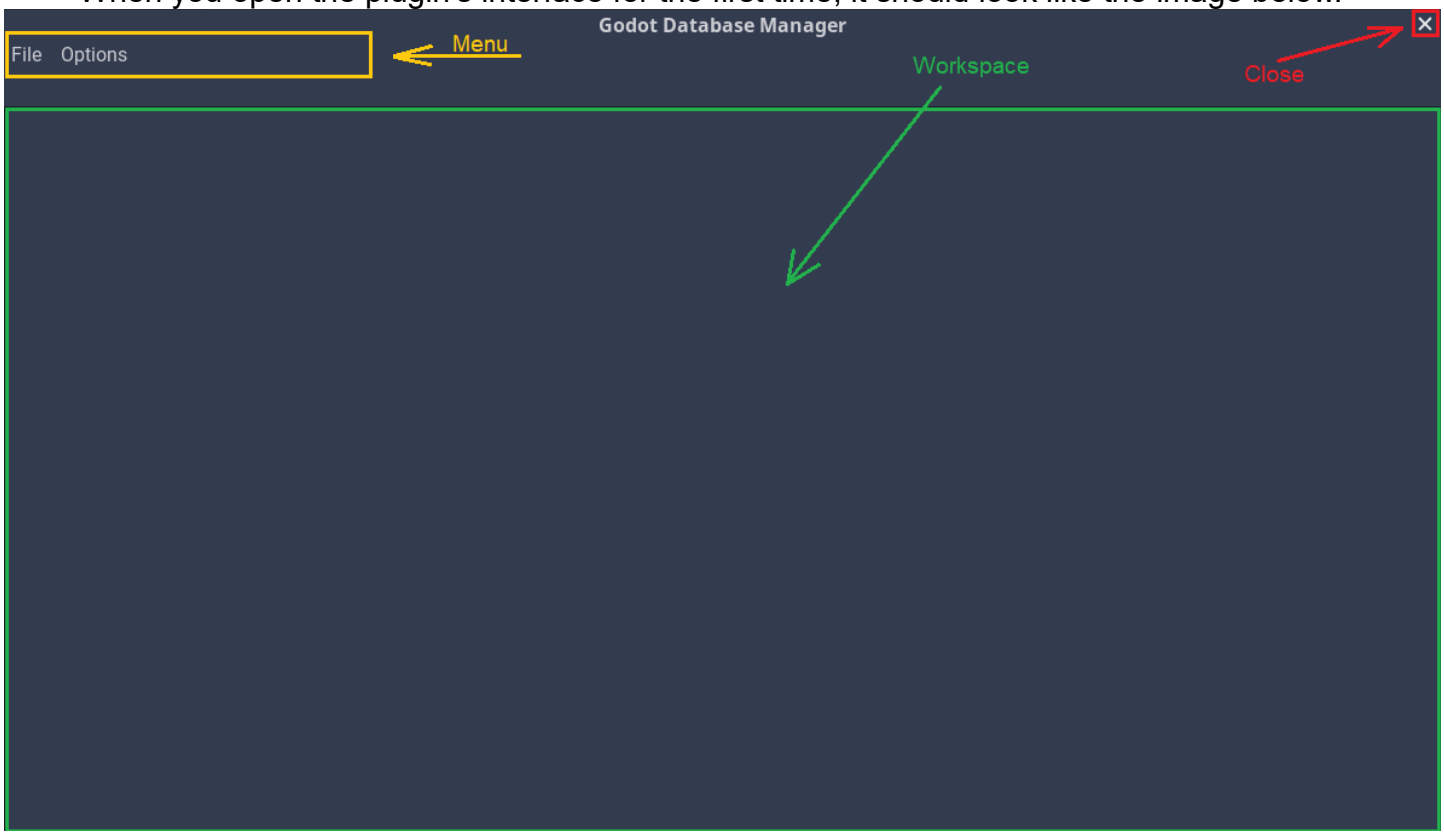


Launching the plugin:



The main interface:

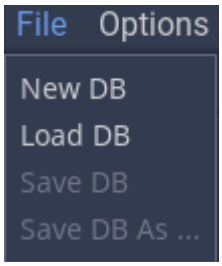
When you open the plugin's interface for the first time, it should look like the image below.



Close button
By clicking on it, will close the plugin's interface.

Menu

◆File menu



– “New DB”: creates a new database
– “Load DB”: loads a database from a JSON file
– “Save DB”: saves the current database to a JSON file
– “Save DB As ...”: saves the current database to a different JSON file

◆Options menu

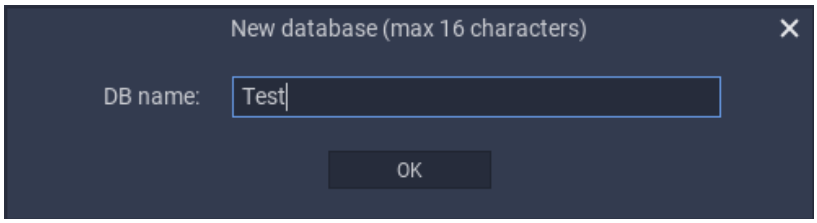
– “Autosave on close”: when enabled, all opened databases will be automatically saved.

Workspace
It is the area where you will edit the databases, tables, properties and data.

Creating a new database:

Choose “File -> New DB”.

Type in the database name and click the “OK” button.



A database name cannot contain the following characters: "`~!@#$%^&*()=+[]{}\|;:'\",<.>/?".

Also there is a limit of 16 characters when choosing a database name.

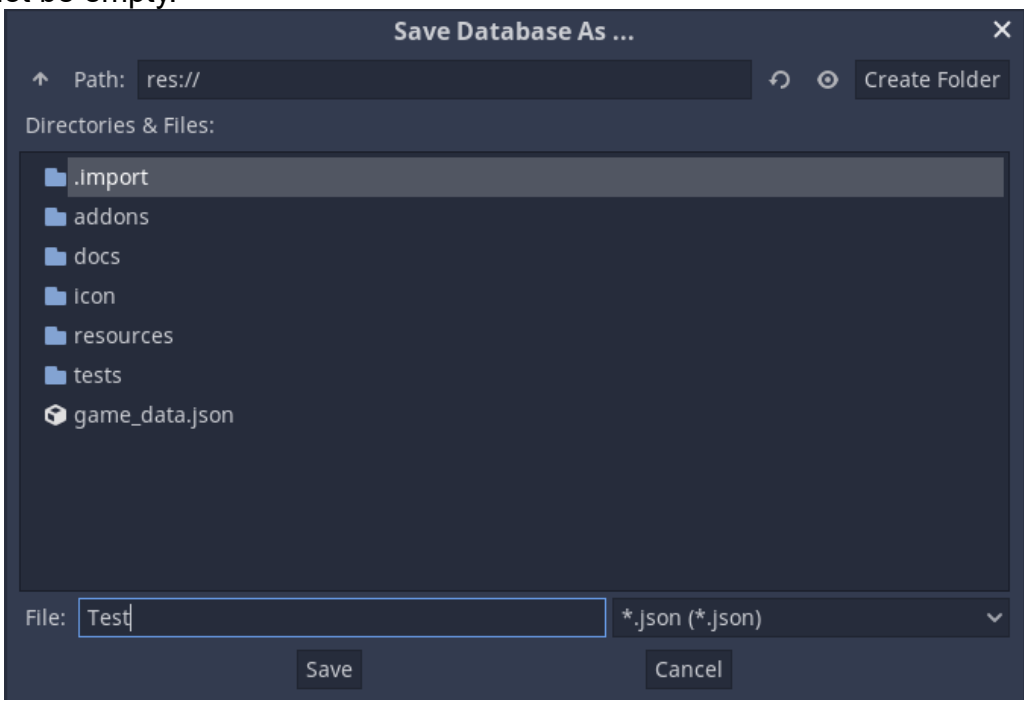
The workspace should look like the image below:

Saving a database:

Choose “File menu -> Save DB”.

Leave the file name as it is, or change it if you want to, and click the “Save” button.

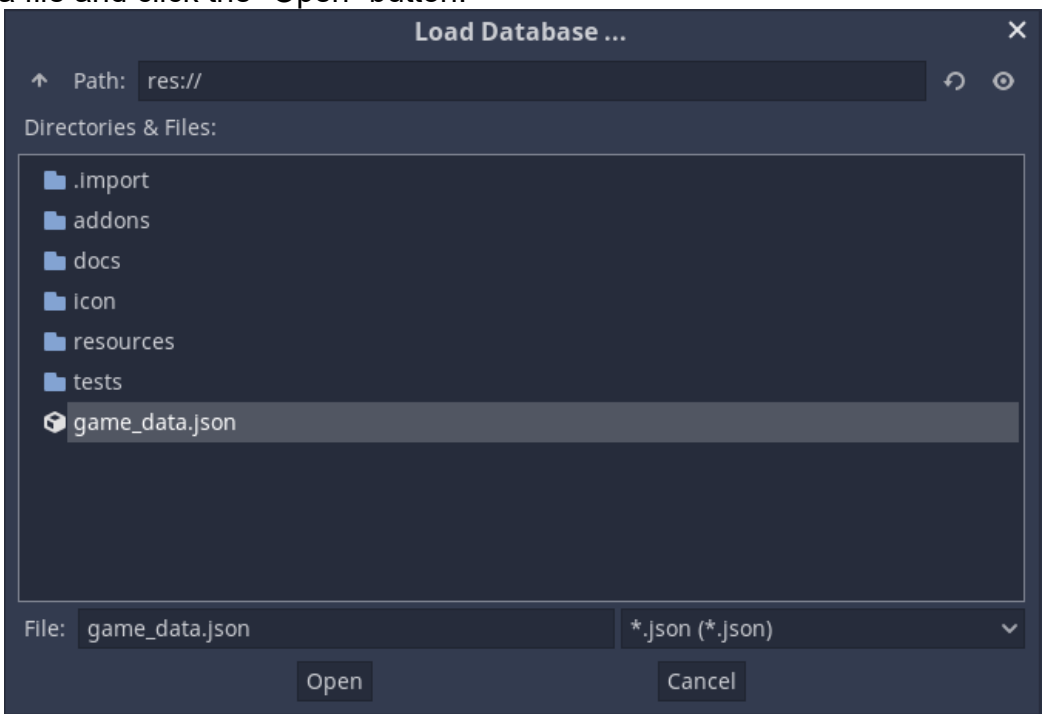
The file name cannot contain the following characters: "`~!@#$%^&*()=+[]{}\|;:'\",<.>/?". Also the file name cannot be empty.



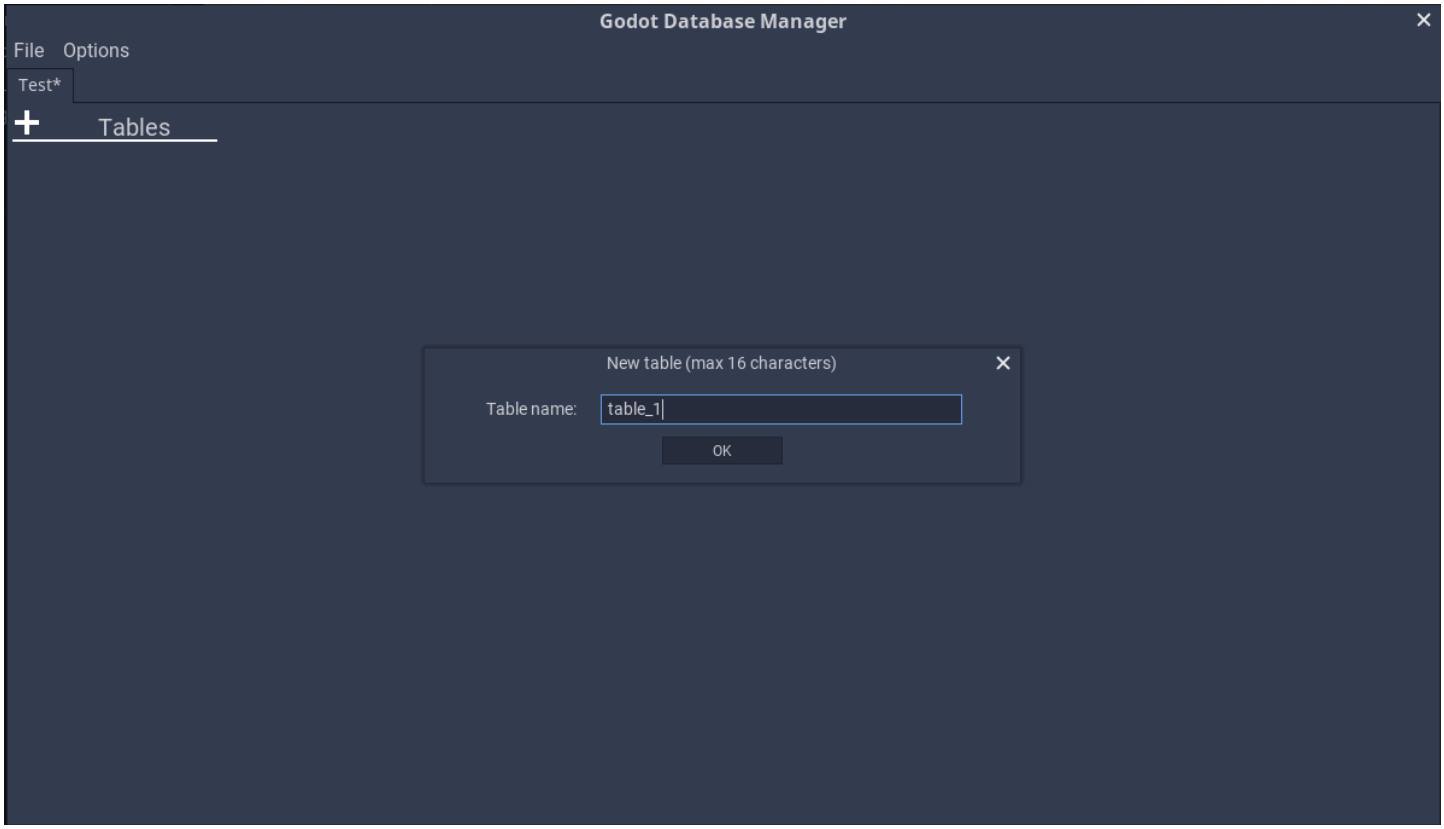
Once a database is saved to a file, that file will be associated to the database and next time when saving it, will not ask you to choose in which file you want to save it.

If you want to choose a different file, choose “File menu -> Save DB As ...”.

Loading a database:
Choose “File menu -> Load DB”.
Choose a file and click the “Open” button.



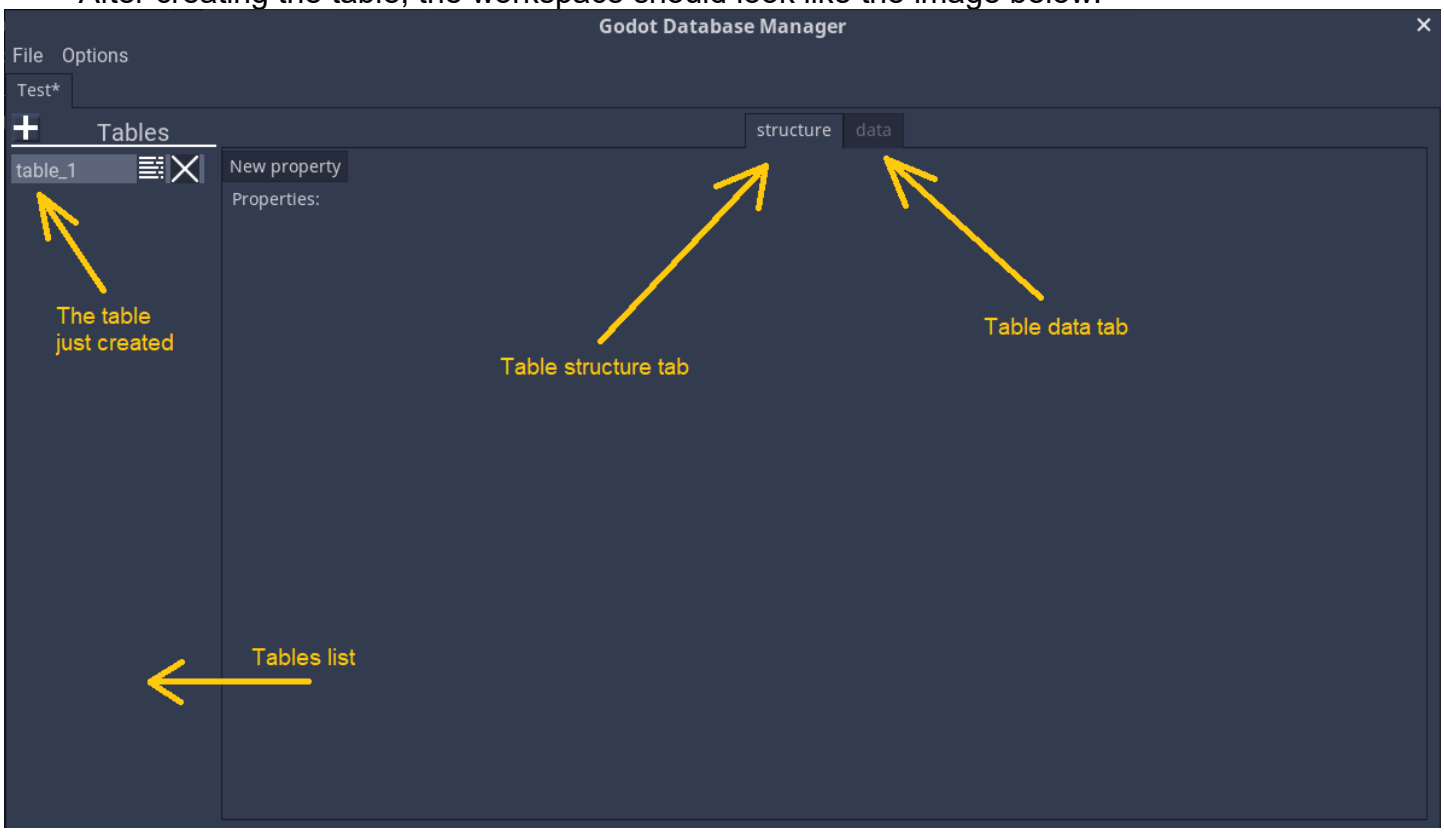
Creating a table and table editor interface:
Press the “+” button. Choose a table name and click the “OK” button.



There is a limit of 16 characters when choosing a table name.

OBS: the table name must be unique in the database.

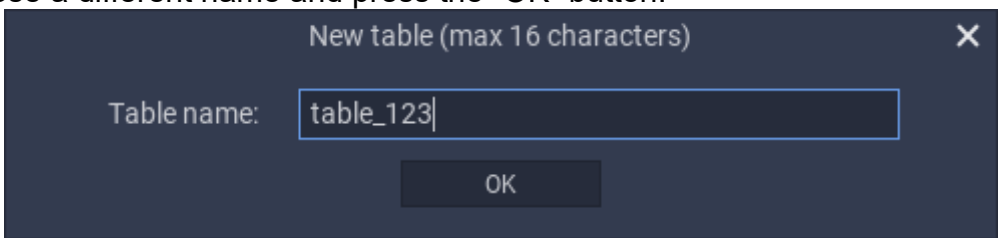
After creating the table, the workspace should look like the image below:



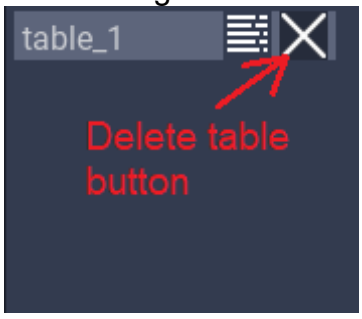
Renaming and deleting tables:

Click the “Edit Table” button. See the image below for more details:

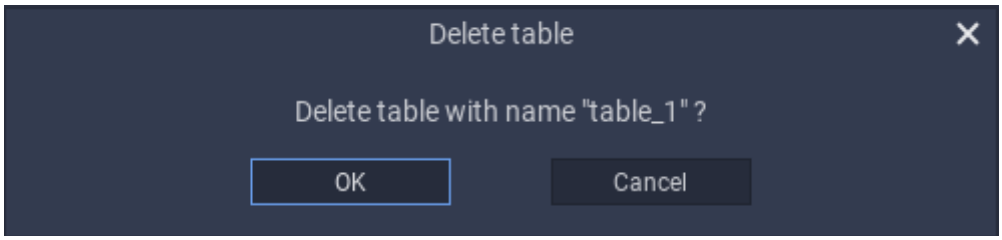
Then choose a different name and press the “OK” button.



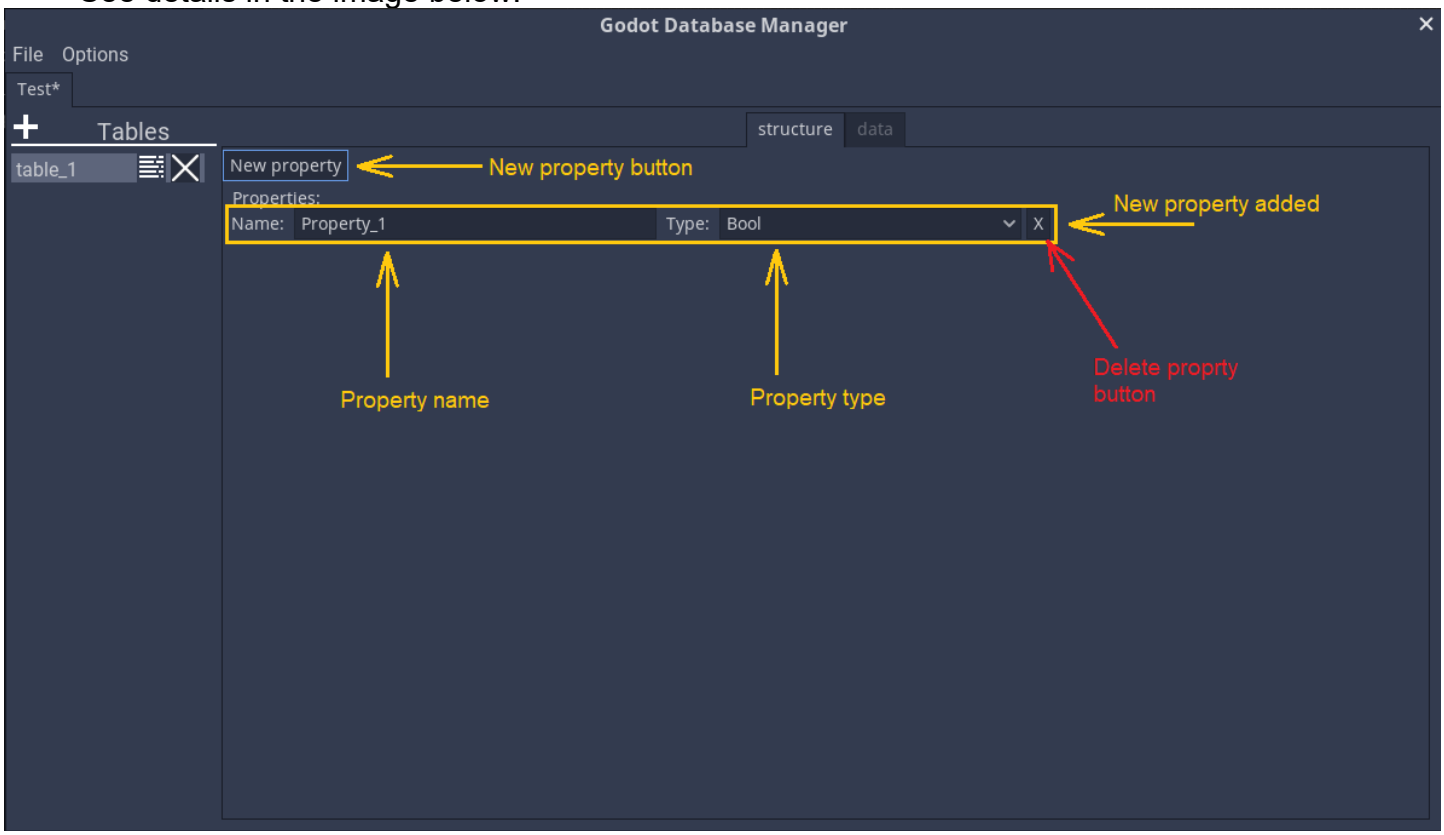
Click the “Delete Table” button. See the image below for more details:



Then confirm the deletion of the table:



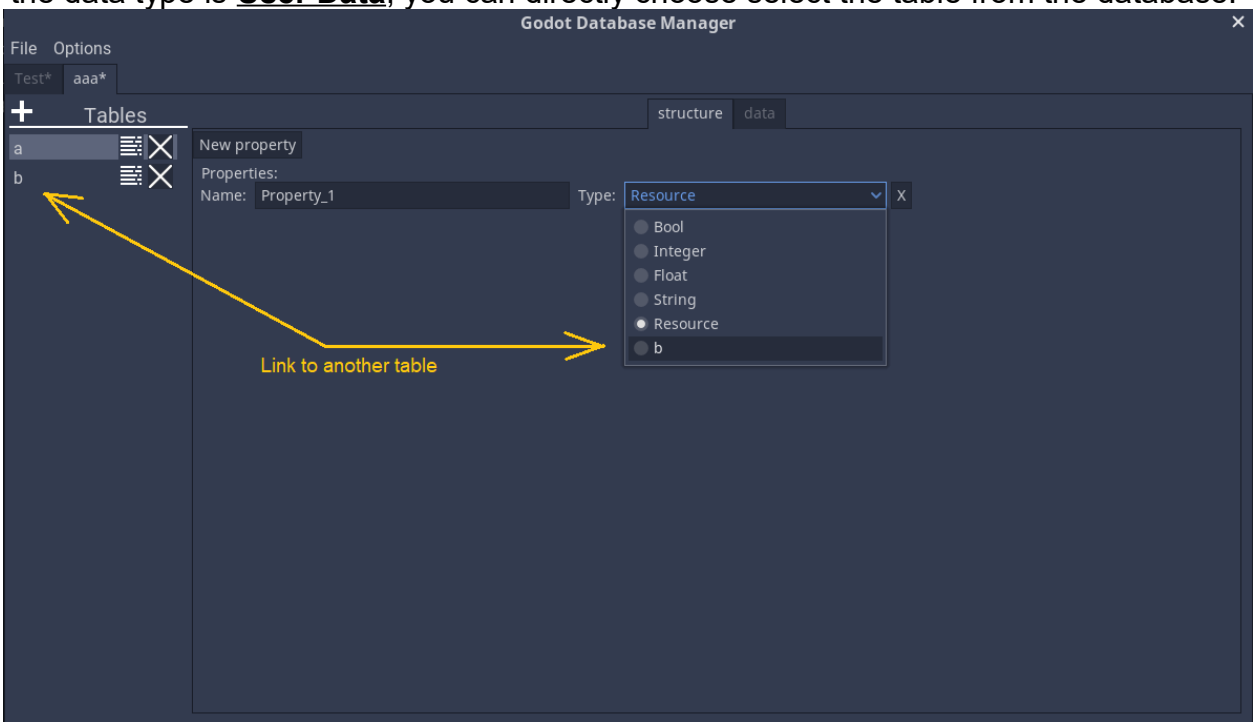
Creating, editing and deleting the properties:
Click on the “New property” button and a new property will automated be added in the table.
After that you can edit the property name and its type.
You can also delete a property by clicking the “Delete property” button.
See details in the image below:



The type of the property can be:
– **Boolean** (Bool): true or false.
– **Integer**: integer number.
– **Float**: floating point number.
– **String**: text.
– **Resource**: a link to a file that is a resource (text, image, sound, video, etc).
– **User data**: a link to another table from the database.

If the type is I**nteger**, you can also choose the option to auto increment the data. The default value is not auto increment.

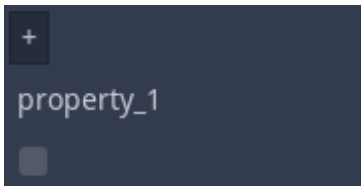
If the data type is **User Data**, you can directly choose select the table from the database.



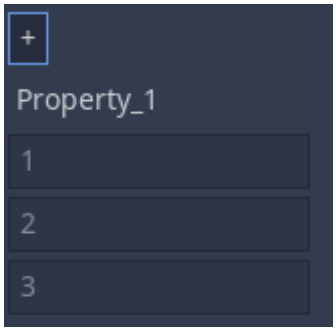
Adding and edit data

Click the “+” button to add data.

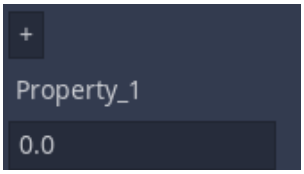
If the data type is **Boolean**, the data is represented by a checkbox and default is false (unchecked).



If the data type is **Integer**, the default value is 0. But if the auto increment option is set to true, the default value is 1 and the data cannot be edited. Also, if you add more data, the next values will be auto incremented.



If the data type is **Float**, the default value is 0.0.

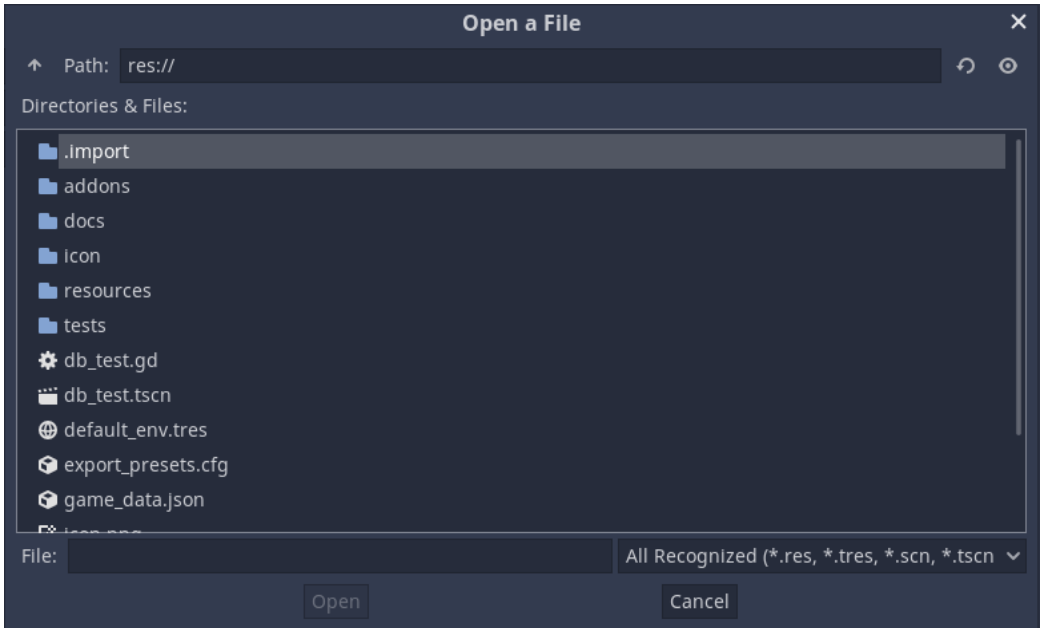


If the data type is **String**, the default data is “” (an empty string). The data can be edited via “Edit data” button or simply editing the text in the LineEdit control.

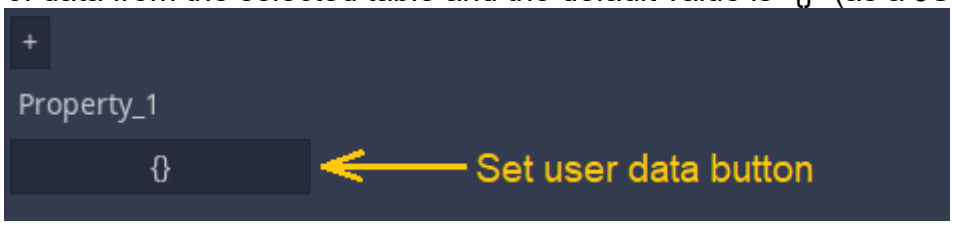
By pressing the “Edit Data” button, a text editor will show up letting you know to edit the data.

If the data type is **Resource**, editing the data is done via a button. The name of the button represents the data and the default value is “res://”.

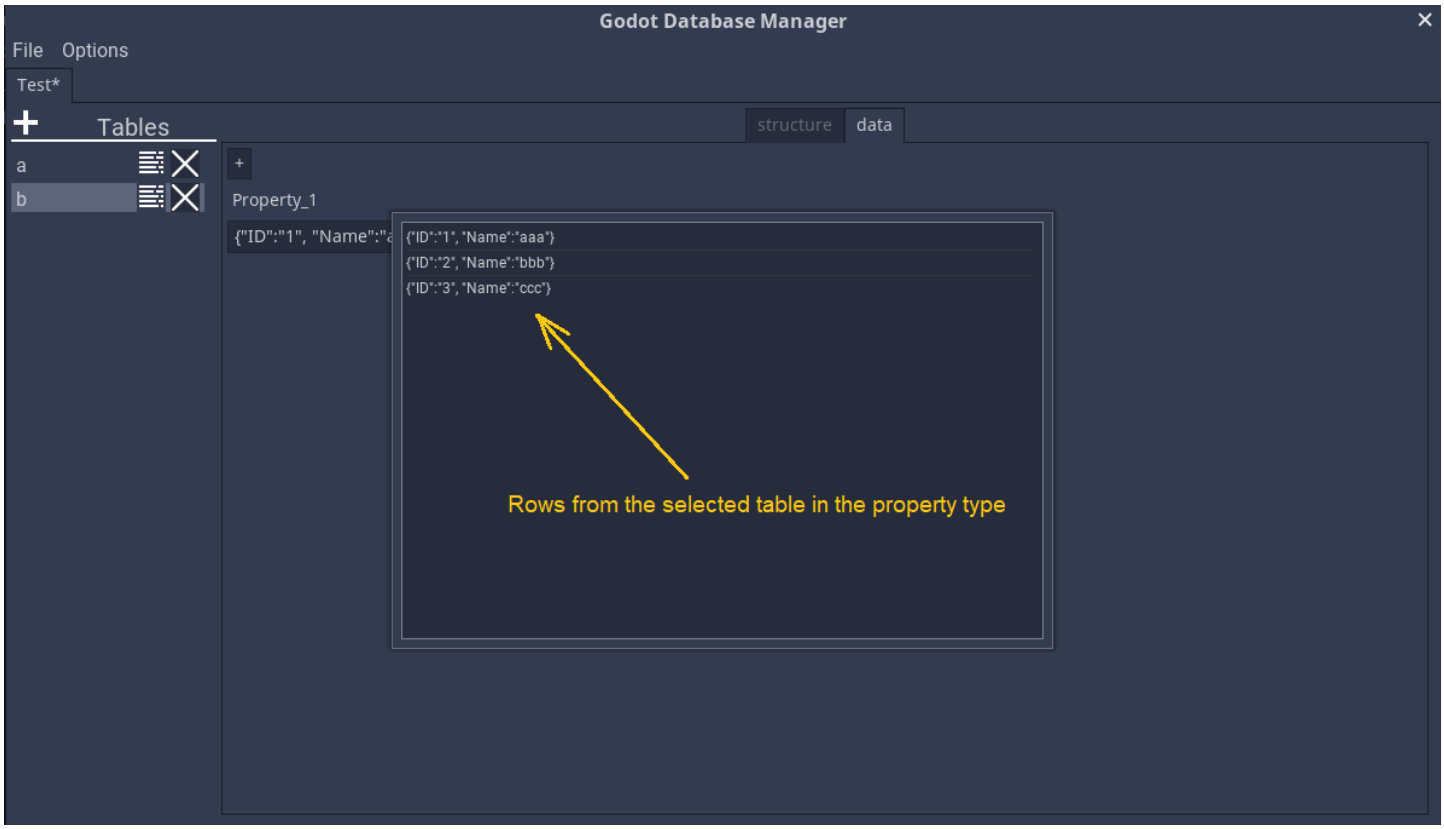
By clicking the button, a dialog will shown up letting you to choose a resource file from the project files.



If the data type is **User Data**, editing the data is done via a button. The name of the button represents a row of data from the selected table and the default value is “{}” (as a JSON).



By clicking the button, a popup will show up letting you to choose a row of data from the selected table.



GDScript files and classes

Constants
File: /addons/godot_db_manager/core/GDDBConstants.gd

c_gddb_signature = “GDDB_ver”
– Godot Database Manager file signature.
– A constant for recognize the GDDB files.
– Type: String

c_gddb_ver = “_current_version_”
– Current API version
– Type: String

c_invalid_id = -1
– A constant used to recognize an integer that is not initialized.
– Type: int

c_max_db_name_len = 16
– A constant that represents the maximum length of a database name.
– Type: int

c_max_table_name_len = 16
– A constant that represents the maximum length of a table name.
– Type: int

c_invalid_characters = "`~!@#$%^&*()=+[]{}\\|;:'\",<.>/?"
– A constant containing all the invalid characters when naming a database.
– Type: String

c_addon_main_path = "res://addons/godot_db_manager/"
– A constant used for easy access the plugin's path.
– Type: String

Types
File: /addons/godot_db_manager/core/GDDBTypes.gd

Database loading errors.
These are returned when loading a database.

e_db_invalid_file – Loading file is not a valid GDDB database file.
e_db_invalid_ver – The version of the loading database file is not the current one.
e_db_valid – The database has been loaded without errors.

Property types.
These are used to recognize the property types.

e_prop_type_bool – Boolean
e_prop_type_int – Integer
e_prop_type_float – Floating point
e_prop_type_string – String
e_prop_type_resource – Resource
e_data_types_count – Used for counting the property types

Data filters.
Used when querying the database for data.

e_data_filter_equal – the value of the data must be equal
e_data_filter_not_equal – the value of the data must be different
e_data_filter_less – the value of the data must be less
e_data_filter_greater – the value of the data must be greater
e_data_filter_lequal – the value of the data must be less or equal
e_data_filter_gequal – the value of the data must be greater or equal

Global functions
File: /addons/godot_db_manager/core/GDDBGlobals.gd

get_data_name(data_type : int) -> String
Description: returns the name of the data type

– Arguments:
  –**data_type**: the type of the data. See **property type** enum for more info.

get_data_filter_name(data_filter_type : int) -> String
Description: returns the name of the data filter

– Arguments:
  –**data_filter_type**: the type of the data filter. See **data filter** enum for more info.

check_db_name(db_name : String) -> bool
Description: checks name of a database. Returns **true** if the name of the database contains valid characters, otherwise **false**. See **c_invalid_characters** constant for more details.

– Arguments:
  –**db_name**: the name of the database

get_json_from_row(table : Object, row_idx : int) -> String
Description: returns a json out of a row data from a table

– Arguments:
  –**table**: the table in the database
  –**row_idx**: the index of the row in the table

get_digits_count(number : int) -> int
Description: returns the count of the digits out of a number

– Arguments:
  –**number**: an arbitrary number

handle_string(text : String) -> String
Description: replace special characters in a string to handle properly saving it into a database.

– Arguments:
  –**text**: the original text

Classes

GDDBMan
Description: Godot database manager
File: /addons/godot_db_manager/core/db_man.gd

Extends: Object

Members:
m_databases – array of databases.

Methods:
add_database(db_name : String) -> int
Description: adds a database. Returns the id of the database just inserted or **c_invalid_id** if the database cannot be added.

– Arguments:
–**db_name**: the name of the database to be added. The name must be unique.

load_database(filepath : String) -> int
Description: loads a database from a file. Returns the id of the database or an error code. See “Database Loading Errors” for more details.

– Arguments:
–**filepath**: the path of the database file.

erase_db_at(idx : int) -> void
Description: deletes a database at a given index.

– Arguments:
–**idx**: the index of the database.

erase_db_by_id(db_id : int) -> void
Description: deletes a database by an id.

– Arguments:
–**db_id**: the id of the database.

erase_db_by_name(db_name : String) -> void
Description: deletes a database by a name.

– Arguments:
–**db_name**: the name of the database.

get_databases_count() -> int
Description: returns the databases count.

get_db_at(idx : int) -> Object
Description: returns a database at a given index.

– Arguments:
  –**idx**: the index of the database.

get_db_by_id(db_id : int) -> Object
Description: returns a database by an id.

– Arguments:
  –**db_id**: the id of the database.

get_db_by_name(db_name : String) -> Object
Description: returns a database by a name.

– Arguments:
  – **db_name**: the name of the database.

generate_new_db_id() -> int
Description: generates a new database id. Internal usage.

can_add_db(db_name : String) -> bool
Description: checks if a database with the same name already exists. Returns **true** if the name is unique, otherwise **false**.
– Arguments:
  – **db_name**: the name of the database.

clear() -> void
Description: clears all databases and the array.

dump(to_console : bool) -> String
Description: dumps the content of the database. Returns a string containing the dump.

– Arguments:
  – **to_console**: if true, the dump will also be added to the debug console.

GDDatabase
Description: Database class.

Extends: Object

Members:
m_db_type – database type. Currently only JSON type is supported.
m_db_id – database id.
m_db_name – database name.
m_tables – array of tables
m_db_filepath – database filepath
m_is_dirty – a flag to check if the database is modified. Used for the interface.

Methods:
set_db_id(db_id : int) -> void
Description: sets the database id.

– Arguments:
–**db_id**: the id of the database.

get_db_id() -> int
Description: returns the id of the database.

set_db_name(db_name : String) -> bool
Description: sets the database name.

– Arguments:
–**db_name**: the name of the database.

get_db_name() -> String
Description: returns the name of the database.

set_db_filepath(filepath : String) -> void
Description: sets the database filepath.

– Arguments:
–**filepath**: the filepath of the database.

get_db_filepath() -> String
Description: returns the filepath of the database.

can_add_table(table_name : String, table_id) -> bool
Description: checks if a table can be added. Returns **true** if the name is unique and the table can be added, otherwise **false**.

– Arguments:
–**table_name**: the name of the table.
–**table_id**: the id of the table. This is used for renaming an existing table.

add_table(table_name : String) -> int

Description: adds a table into the database. Returns the table id if the table is successfully added or **c_invalid_id** if a table with the same name already exists.

– Arguments:

–**table_name**: the name of the table.

edit_table_name(table_name : String, table_id : int) -> bool

Description: renames a table. If the name is used by another table returns **false**, otherwise **true**.

– Arguments:

–**table_name**: the name of the table.

–**table_id**: the id of the table.

delete_table_at(idx : int) -> void

Description: deletes a table at a given index.

– Arguments:

–**idx**: the index of the table.

delete_table_by_id(table_id: int) -> void

Description: deletes a table by an id.

– Arguments:

–**table_id**: the id of the table.

delete_table_by_name(table_name: String) -> void

Description: deletes a table by a name.

– Arguments:

–**table_name**: the name of the table.

generate_new_table_id() -> int

Description: generates a new table id. Internal usage.

get_tables_count() -> int

Description: returns the tables count

is_table_exists(table_name : String) -> bool

Description: returns **true** is a table with the name exists, otherwise **false**.

– Arguments:

–**table_name**: the name of the table.

get_table_at(idx: int) -> Object
Description: returns a table at a given index.

– Arguments:
  –**idx**: the index of the table.

get_table_by_id(table_id: int) -> Object
Description: returns a table by an id.

– Arguments:
  –**table_id**: the id of the table.

get_table_by_name(table_name: String) -> Object
Description: returns a table by a name.

– Arguments:
  –**table_name**: the name of the table.

clear() -> void
Description: clears the database content.

set_dirty(dirty : bool) -> void
Description: sets a flag if the database is modified.

– Arguments:
  –**dirty**: if true, than the database is modified.

is_dirty() -> bool
Description: returns **true** if a database has been modified, otherwise **false**.

save_db() -> void
Description: saves a database.

load_db() -> int
Description: loads a database. Returns e_db_valid if the database is successfully loaded or an error code. See “Database Loading Errors” for more details.

dump() -> String
Description: dumps the content of the database. Returns a string containing the dump.

GDDBTable
Description: Table class.
File: /addons/godot_db_manager/core/db_table.gd

Extends: Object

Members:
m_table_id – table id
m_table_name – table name
m_props – array of properties
m_data – array of data
m_rows_count – rows count
m_parent_database – parent database

Methods:
set_table_id(table_id : int) -> void
Description: sets the table id.

– Arguments:
–**table_id**: the id of the table.

get_table_id() -> int
Description: returns the id of the table.

set_table_name(table_name: String) -> void
Description: sets the table name.

– Arguments:
–**table_name**: the name of the table.

get_table_name() -> String
Description: returns the name of the table.

set_parent_database(db : Object) -> void
Description: sets the parent database.

– Arguments:
–**db**: database.

get_parent_database() -> Object
Description: returns the parent database.

add_prop(prop_type : int, prop_name : String) -> int
Description: adds a property.

– Arguments:
–**prop_type**: the type of the property.
–**prop_name**: the name of the property.

add_table_prop(prop_name : String, table_name : String) -> int
Description: adds a property as a table type. Returns the property id.

– Arguments:
  –**prop_name**: the name of the property.
  –**table_name**: the name of the table.

link_tables_props() -> void
Description: links custom properties from tables. Internal usage.

edit_prop(prop_id : int, prop_type : int, prop_name: String) -> void
Description: edits a property.

– Arguments:
  –**prop_id**: the id of the property.
  –**prop_type**: the type of the property.
  –**prop_name**: the name of the property.

enable_prop_autoincrement(prop_id : int, enable : bool) -> void
Description: enables or disables auto increment option to a property.

– Arguments:
  –**prop_id**: the id of the property.
  –**enable**: enable/disable flag.

delete_prop(prop_id : int) -> void
Description: deletes a property by an id.

– Arguments:
  –**prop_id**: the id of the property.

generate_new_prop_id() -> int
Description: generates a new property id. Internal usage.

get_props_count() -> int
Description: returns the properties count.

get_prop_at(idx : int) -> Object
Description: returns a property at a given index.

– Arguments:
  –**prop_idx**: the index of the property.

get_prop_by_id(prop_id : int) -> Object
Description: returns a property by an id.

– Arguments:
–**prop_id**: the id of the property.

add_blank_row() -> void
Description: adds a blank row of data.

add_row(data_array : Array) -> void
Description: adds a row of data.

– Arguments:
–**data_array**: the array with data.

remove_row(row_idx : int) -> void
Description: deletes a row of data at a given index.

– Arguments:
–**row_idx**: the index of the row.

get_rows_count() -> int
Description: returns the rows count.

edit_data(prop_id : int, row_idx : int, data : String) -> void
Description: edits a data.

– Arguments:
–**prop_id**: the id of the property.
–**row_idx**: the index of the row.
–**data**: the new data.

get_data_size() -> int
Description: returns the amount of data.

get_all_data() -> Array
Description: returns the data array.

get_data_at(idx : int) -> String
Description: returns a data by a given index.

– Arguments:
–**idx**: the index of the data.

get_data(prop_id : int, row_idx : int) -> String
Description: returns a data by a property id and at a given row index.

– Arguments:
  –**prop_id**: the id of the property.
  –**row_idx**: the index of the row.

get_data_at_row_idx(row_idx : int) -> Array
Description: returns an array of data at a given row index.

– Arguments:
  –**row_idx**: the index of the row.

get_data_by_prop_id(prop_id : int, data_filter : int) -> Array
Description: returns an array of daya by a property id and using a filter.

– Arguments:
  –**prop_id**: the id of the property.
  –**data_filter**: the filter type of the data. See data filters for more details.

get_data_by_prop_name(prop_name : String) -> Array
Description: returns an array of data by a property name

– Arguments:
  –**prop_name**: the name of the property.

get_data_by_data(data_value : String, data_filter : int) -> Array
Description: returns an array of data filtered by data and data filter type.

– Arguments:
  –**data_value**: the value of the data.
  –**data_filter**: the data filter type. See data filters for more details.

get_data_by_prop_name_and_data(prop_name : String, data_value : String) -> Array
Description: returns an array of data filtered by a property name and a data value.

– Arguments:
  –**prop_name**: the name of the property.
  –**data_value**: the value of the data.

– Example:
– get_data_by_prop_name_and_data(“id”, “1”) => will return a row of data where the “id” is “1”.

clear() -> void
Description: clears the table's content.

dump() -> String
Description: dumps the content of the table. Returns a string containing the dump.