PLATINUMSOFTWARE

Grape SQL Function Reference

Table of Contents

1. GRAPE SQL FUNCTIONS

1.1 API result functions

These functions deals with the creation of standardized API results (in JSON format) to be sent back to the API call. They can be found in api_result_json.sql

NAME	PARAMETERS	DESCRIPTION
grape_result_type	*success* BOOLEAN *reason* TEXT *data* JSON	Grape result types.
api_result_error	*message* TEXT *code* INTEGER *error* JSON	Returns a standardized JSON error object with status as "ERROR" and the other fields populated. Example: `{"status": "ERROR", "message": "Message", "code": -2, "error": {} }`
api_error	*message* TEXT *code* INTEGER *error* JSON	Overload for *api_result_error*.
api_error		With no arguments, an "Unknown error" message will be generated. Example: `{"Unknown error", -1}`
api_error_invalid_input	*info* JSON	Similar to calling `api_result_error("Invalid input", -3)`
api_error_invalid_field	*name* TEXT	Similar to calling `api_result_error("Missing or invalid field: ", -3)`
api_error_permission_denied	*info* JSON	Similar to calling `api_result_error("Permission denied", -2)`
api_error_data_not_found	*info* JSON	Similar to calling `api_result_error("Data not found", -5)`
api_error_invalid_data_state	*info* JSON	Similar to calling `api_result_error("The operation requested could not be performed on the data because the data is not in a valid state", -6)`
api_success	*keys* TEXT[] *values* TEXT[] *types* TEXT[]	This function will construct a JSON object containing at least one field, "status" with the value "OK". The 3 input parameters should be arrays containing additional keys, values and the associated types *(n/i/number/integer, j/json or nothing for text)*.
api_success	*keys* TEXT *values* INTEGER	
api_success	*key1* TEXT *val1* INTEGER	Create an API result success JSON object with one integer field added.
api_success	*key1* TEXT *val1* INTEGER *key2* TEXT *val2* INTEGER	Create an API result success JSON object with two integer fields added.
api_success	*key1* TEXT *val1* JSON	Create an API result success JSON object with a JSON field merged into the result.
api_success		Returns a API result object with a "status" field set to "OK".

NAME	PARAMETERS	DESCRIPTION
api_success_if_not_null	***fieldname*** TEXT ***data*** JSON	Returns success message when data is ***NOT NULL***, otherwise it returns `grape.api_error_data_not_found()`
api_result	***res*** grape_result_type	Returns error message similar to calling `api_error(res.reason, -1)` if false, otherwise returns success message similar to calling `api_success("data", res.data)`

1.2 Data importing functions

These functions deals with how data importing is handled. They can be found in data_import.sql

NAME	PARAMETERS	DESCRIPTION
upsert_data_import_type	***processing_function*** TEXT ***short_description*** TEXT ***file_format_info*** TEXT ***function_schema*** TEXT ***param_definition*** JSON	Upsert data import types. If processing_function name is the same, all other values are updated.
estimate_datatype		Overloaded function to estimate the potential datatype of a text value. Function returns the data_type. Example: `{"NULL", "INTEGER", "NUMERIC", "DATE", "TIMESTAMP", "TIMESTAMPTZ"}`
data_import_insert		API function to insert a data_import entry. Returns success message similiar to calling `api_success("data_import_id")`
data_import_delete		API function to delete a data_import entry. Returns success message similiar to calling `api_success()`
data_import_row_insert		API function to insert a row of JSON into data_import_row. Required field data_import_id must be in the JSON data.
data_import_done		API function to notify server that insertion of all the rows has been completed and timestamp this completion. Returns message similiar to calling `api_success("data_import_id")`

data_import_process	***data_import_id*** INTEGER	Internal function to process data_import data. Returns message based on the following data import status:
		`0 - Empty`
		`1 - Populated`
		`2 - Process started`
		`3 - Some not processed`
		`4 - Processed`

data_import_process		API function to process data_import data. Calls internal process function. Returns error message similar to calling `api_error("data_import_process failed", -1)` if false, otherwise returns success message similar to calling `api_success()`
data_import_test_table_insert		API function to create a test table from data_imports data.
data_import_test_table_drop		API function to drop a test table from data_imports data.

NAME	PARAMETERS	DESCRIPTION
data_import_build_result	***status*** TEXT	Builds an object in the form of `{"result": {"status": "OK"}` for returning from data import functions.
data_import_build_result	***status*** TEXT ***shared_data*** JSON	Builds a object in the form of `{"result": {"status": "OK"}, "shared_data": {}}` for returning from data import functions.
data_import_build_result	***status*** TEXT ***shared_data*** JSONB	Builds a object in the form of `{"result": {"status": "OK"}, "shared_data": {}}` for returning from data import functions.
data_import_reset	***data_import_id*** INTEGER	Resets data_import_id's data import status to 1, if populated.
data_import_test_table_alter		API function that returns message similiar to calling `api_success()`
dimport_generic	***data_import*** grape.data_import ***args*** JSONB	Example dimport function that does not process the data in any way and allows for a way to create a test table with data that does not need to be processed. Returns message similiar to calling `data_import_build_result("OK")`
upsert_data_import_type		This function does not actually process the data in any way, but is a way to allow you to import data with which you may create test tables in grape.
proc_process_data_import		Process to process data import files in the background via ps_bgworker.

1.3 JSON helpers

These functions are JSON helpers and can be found in several files: json2xml.sql, json_diff.sql, json_to_composite_type_text.sql, cast_json_array_to_int_array.sql, cast_json_array_to_text_array.sql

NAME	PARAMETERS	DESCRIPTION
json2xml	***data*** JSON ***root*** TEXT	Converts JSON object to xml.
json_diff	***old*** JSONB ***new*** JSONB	Compares two JSON objects and returns an object containing fields that are different between the two objects. If a field exists in **j_old**, but not in **j_new**, it is not included in the results. If a field exists in **j_new**, but not in **j_old**, it is included in the results. If a field is different, **j_new** is chosen.
json_diff	***old*** JSON ***new*** JSON	
json_object_diff	***old*** JSONB ***new*** JSONB	Compares two JSON objects and return any values that exists in **_new** but not in **_old**.
json_array_diff	***old*** JSONB ***new*** JSONB	Compare two JSON arrays and return any values that exists in **_new** but not in **_old**.
json_to_composite_type_text	***target_schema*** TEXT ***target_type*** TEXT ***data*** JSON	Converts JSON object to composite type text.
json_to_composite_type	***target_schema*** TEXT ***target_type*** TEXT ***data*** JSON	This function will populate a custom type from a JSON object. Multi-level nested objects are supported.
cast_json_array_to_int_array	***JSON*** data	Provides an implicit cast from JSON to ***INT[]*** (cast_json_array_to_int_array.sql).

NAME	PARAMETERS	DESCRIPTION

		`#select cast_json_array_to_int_array('[1,2,3]'::JSON);`
		`cast_json_array_to_int_array`
		`------------------------------`
		`{1,2,3}`

cast_json_array_to_text_array	*JSON* data	Provides an implicit cast from JSON to *TEXT[]* (cast_json_array_to_text_array.sql).
		`#select cast_json_array_to_text_array('[aa,bb,cc]'::JSON);`
		`cast_json_array_to_text_array`
		`------------------------------`
		`{"aa","bb","cc"}`

1.4 List query

Grape's list_query call provides an easy way to retrieve rows from a table. Before the contents of a table can be retrieved this way it needs to be added to a whitelist. This functions can be found in list_query.sql. The built-in API call to access this function is **/grape/list**. Access control is enforced on tables retrieved.

The **grape.list_query** function returns rows from a database table. The following input fields are recognized:

- tablename TEXT
- schema (optional) TEXT
- sortfield (optional) TEXT
- sortorder (optional) TEXT DESC
- limit (optional) INTEGER (DEFAULT 50)
- offset (optional) INTEGER (DEFAULT 0)
- filter (optional) array of fields:
  - field TEXT
  - operand TEXT of '=', '>', '<', '>=', '<=', 'LIKE', 'ILIKE', 'IS_NULL', 'IS_NOT_NULL', 'IN'
  - value TEXT

The following functions deals with the access control:

NAME	PARAMETERS	DESCRIPTION
grape.list_query_whitelist_add	*schema* TEXT *tables* TEXT[] - A list of table names to allow *roles* TEXT[] - A list of roles to allow	Adds tables to the whitelist for use in grape list_query. Users must be in _roles to be able to access the data in the table.
grape.list_query_whitelist_delete	*schema* TEXT *tablename* TEXT - A table to remove from allow	Removes a table from the whitelist.
grape.list_query_check_permission	*schema* TEXT *tablename* TEXT	Check permission on a table for current user.

1.5 Table Operations

Grape provides three API calls to perform generic DML (INSERT, UPDATE and DELETE) on whitelisted tables.

The API calls are:

- GrapeInsertRecord
- GrapeUpdateRecord
- GrapeDeleteRecord

The SQL function used to whitelist tables, is:
`grape.table_operation_whitelist_add(schema TEXT, tables TEXT[], roles TEXT[], allowed_operation TEXT)` .

- *schema* - The schema of the table
- *tables* - An array of table names to add
- *roles* - An array of role names to allow
- *allowed_operation* - The operation to allow (INSERT, UPDATE or DELETE)

1.6 Reports

These functions can be found in reports.sql

NAME	PARAMETERS	DESCRIPTION
save_report	*report_id* INTEGER *name* TEXT *description* TEXT *function_schema* TEXT *function_name* TEXT *input_fields* JSON	
save_report	*name* TEXT *function_name* TEXT *description* TEXT *input_fields* JSON	
save_report	*settings* JSON	
execute_report	*report_id* INTEGER *parameters* JSON	
execute_report	*parameters* JSON	JSON object needs name field (with report name) and optional JSON parameters.
execute_report_to_file	*report_id* INTEGER *reports_executed_id* INTEGER *parameters* JSON	Function to convert a report to a file.

1.7 User related functions

NAME	PARAMETERS	DESCRIPTION
grape.user_save	JSON containing: *user_id INTEGER* *username TEXT* *password TEXT* *email TEXT* *fullnames TEXT* *active BOOLEAN (optional)* *role_names TEXT[]* *employee_guid* UUID	Save a user field, or create a new user. API call: `POST /grape/user/save`

NAME	PARAMETERS	DESCRIPTION
grape.new_user	***user_id*** INTEGER ***rec*** RECORD ***role_name*** TEXT	Creates a new user. Returns the user ID, or error code -1 if it does not exist.
grape.username	***user_id*** INTEGER	Returns the username for a user ID, or ***NULL*** if it does not exist.
grape.user_id_from_name	***username*** TEXT	Returns the user ID for a username, or ***NULL*** if it does not exist.
grape.user_id_from_fullnames	***fullnames*** TEXT	Returns the user ID for a user found by fullnames, or ***NULL*** if it does not exist.
grape.username_from_fullnames	***fullnames*** TEXT	Returns the username for a user found by fullnames, or ***NULL*** if it does not exist.
grape.hash_user_password	***user_id*** INTEGER	Hashes a password for user and updates the user table afterwards. 1. If the hash length is the same as the password length and the password starts with a '$' sign, it is assumed that the password is already hashed and the update is ignored (return -1) 2. If grape.setting passwords_hashed isn't true, nothing is done (return -2) 3. On success 0 is returned
grape.hash_user_password	***username*** TEXT	Overload for ***grape.hash_user_password (user_id INTEGER)***
grape.set_user_password	***user_id*** INTEGER ***password*** TEXT ***is_hashed*** BOOLEAN	Set user password. If the password given to this function is already hashed then ***is_hashed*** should be ***TRUE***.

1.8 Session related functions

NAME	PARAMETERS	DESCRIPTION
grape.current_user_roles		Returns a list of all roles the current user belongs to.
grape.current_user_in_role	***role*** TEXT	Returns ***TRUE*** if the current user belongs to ***_role***.
grape.current_user_in_role	***roles*** TEXT[]	Returns ***TRUE*** if the current user belongs to any of ***_roles***.
grape.current_user_id		Returns the integer value of the current session's ***"grape.user_id"*** setting. This is typically set with grape before any API call is called.
grape.check_session_access	***session_id*** TEXT - Session ID to check for ***check_path*** TEXT - Access path to check ***check_method*** TEXT - HTTP method to check (GET/POST)	This function performs access control on an API call (based on the path and session ID). It is automatically called by the express app before any API call is performed: 1. Check that the path has access control on it. If it cannot be found, the grape setting default_access_allowed is checked, and if true, access will be granted. If not, it will be denied and code 9 will be returned. 2. If the path has a role 'guest' granted access to it, everyone will be allowed (even if the session is invalid). 3. If the session is invalid, access will be denied and code 1 returned. 4. If the path has a role 'all', only, and all, valid sessions will be granted access.

NAME	PARAMETERS	DESCRIPTION
		5. If the user has access granted to the access path's role, access is granted. 6. If all the above fails, access is denied with code 2.
grape.session_insert	***user_id*** INTEGER ***ip_address*** TEXT	This function requires user.sql. 1. Input: • username or email • password • ip_address • persistant TRUE/FALSE (optional) 2. Status: • status = ERROR • code 1 = No such user • code 2 = Wrong password • code 3 = User is inactice • code 4 = IP not allowed 3. On success, status = OK and following fields returend: ***session_id***, ***user_id***, ***username*** and ***user_roles***. 4. Setting ***hash_passwords*** is used to decide if passwords are hashed or not.
grape.logout	***JSON*** JSON containing session_id	API call: `/grape/logout`
grape.session_ping	***JSON*** JSON containing session_id	Checks validity of a session and returns a JSON object containing the session's username, user_id, fullnames, email, GUID and user_roles. API call: `/grape/session_ping`
grape.session_insert	***username*** TEXT ***password*** TEXT	This function inserts a new session for a valid username and password provided. API call: `/grape/logout`
grape.create_session_from_service_ticket		Function which creates a session from the service ticket.
grape.set_password_with_service_ticket		Function which sets the password with the service ticket.
grape.logout		Function used to logout user from current session.
grape.pint		Function used to ping current session.
grape.set_session_user_id	***user_id*** INTEGER	Function which sets the ***user_id*** for the current session.
grape.set_session_username	***username*** TEXT	Function which sets the ***user_name*** for the current session.

1.9 Other utility functions

NAME	PARAMETERS	DESCRIPTION
month_diff	***_d1*** DATE ***_d2*** DATE	Returns an integer containing the number of months between the two dates provided. If the first parameter is after the second (higher date), the return value will be negative.
set_value	***_name*** TEXT ***_value*** TEXT	Sets the value (insert if new, replace if exist) in the key-value pair table grape.setting returning _value.

NAME	PARAMETERS	DESCRIPTION
get_value	***_name*** TEXT ***_default_value*** TEXT	Gets the value for setting ***_name***, and if not found it will return ***_default_value***. Defined in setting.sql
generate_uuid		Generates a unique UUID (for example b1086d35-e973-4356-3adc-2eeb6f4963e2). Defined in uuid.sql
array_lowercase	TEXT[]	
clean_telephone_number	***_tel*** TEXT	Returns a text containing only numbers.
random_string	***length*** INTEGER	Generates a random string of ***length*** length. Defined in random_string.sql