RESTful SQL API Documentation

CONNECT - Connect to a database

POST /connect (connect to a database)

Body:
{

host*	string *example: localhost* The server name
port*	integer *example: 3306* The database port
username*	string *example: root* Username
password*	string *example: password* Password
database*	string *example: database* The database name

}

Reponse:
200 OK
401 Failed to connect to the database

Explanation:
Connect to a local or remote database by passing in all the required information. A successful connection is required to use any of the API endpoints.

Assumption:
The user has created a database before using theAPI.

Limitation:
Create database is not supported currently.

TABLE - Manage tables

POST /table (create a new table in the database)

Body:
{

name* string
example: NewTable
The name of the table to be created

columns* string
example: col1,col2,col3,col4

A list of columns in this new table, separate by comma

uniques string
example: col1,(col2,col4)

A list of columns that have unique key on it/them, separate by comma, and composite key is grouped by parentheses

}

Return:
201 Created
400 Bad Request
401 Unauthorized access
412 Invalid arguments

Explanation:
Create a new table with specified name and list of columns and unique keys(indexes). As default, all the columns will be set to varchar(200) as their default type. The list of column names is separated by comma, and there *SHOULD NOT* have space at any point in this list. This is the same for the list of indexes, each index is separated from others by comma, and for composite indexes, parentheses should be used to group all the elements for a composite index.

Assumption:
Here are some pre-conditions when to use this function:
- The table name must not exist in the database before, to check this assumption, please go to the GET Metadata function and query by ‘TABLE’ to make sure the new table name is not in the result.
- There should not be any duplicate columns in the columns field.

- All elements that appear in a unique field must also appear in the column field.

Limitation:
For this function, whatever errors occur during the executing time, the whole process would be aborted. Hence, a very small mistake on input can cause the whole function to fail. This can make sure the schema fits the users' needs, but it causes some inconvenience.

PUT /table (insert or remove columns from an existing table)

Body:
{

name* string
example: Table1
An existing table name

columns* string
example: Column1, Column2, Column3
A list of column names

operation* string
example: insert
Operation mode: insert, drop. If the mode is inserted, the columns will be added to the table. If the mode is drop, the columns will be removed from the table
Enum:
[insert, drop]

}

Reponse:
200 OK
400 Invalid Operation
401 Unauthorized access

Explanation:
Insert or remove table columns by specifying the column names in a comma separated list. The data type of the new insert column is VARCHAR(200) by default.

Assumption:
The table must exist in the database. To insert a column, the column name does not exist in the table. To remove a column, the column name exists in the table.

Limitation:
The default data type is VARCHAR(200), but the data type can be changed using the UPDATE /metadata endpoint.

DELETE /table/{table_name} (delete an existing table from the database)

Params:

table_name*	string *example: table1* An existing table name.

Response:

200 OK
400 The table does not exist in the database
401 Unauthorized access

Explanation:
Delete all the data inside of an existing table and remove the table itself.

Assumption:
The table already exists in the database.

TABLE/DATA - Manage data records

PUT /table/data (update records for one selected table)

Body:
{

name*	string *example: table1* The table name
columns*	string *example: table1.id* The row names
values*	string *example: 4440* User defined value of selected row
condition	string *example: table1_id=table2_id* The conditions needed to match

}

Reponse:
201 Created;
400 Bad Request
401 Unauthorized access;
412 Invalid arguments

Explanation:
This function updates selected records(row) for one selected table.

Assumption:
There are some pre-condition of this function. The first requirement is that the name must exist in the database. Moreover, all the specified columns must be in that table, and there should not be any duplicate columns in the parameter. The number of tables should only be one.

Limitation:
The advanced version has not yet been completed.

GET /table/data (get the data from an existing table)

Params:

name*	string *example: table1* An existing table name.
columns	string *example: table1.id* Specify the column to retrieve. All columns are returned by default.
page	string *example: 1* Each page returns 250 rows. Setting the page number can retrieve more data and the default page is 1.
filter	string *example: <=* Extract only those records that fulfill the filter condition. It supports operators: =, >, <, >=, <=, !=, BETWEEN, LIKE, and IN. It can be combined with AND, OR, and NOT operators. An example: column1 = 1 OR column2 = 2
sort_by	string *example: ASEC* Sort the result set in ascending or descending order. The sort_by keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword. An example: column1 ASEC, column2 DESC

Reponse:

200 OK
400 Table does not exist in the database
401 Unauthorized access

Explanation:
Get the data from an existing table in the database.

Assumption:
The table exists in the database.

Limitation:
This operation doesn’t support complex aggregation such as sort, avg, min, and max. Please check POST /groupby for advanced aggregation.

DELETE /table/data (delete records of a single table with pre-conditions)

Body:
{

name*	string *example: table1* The table name
condition	string *example: table1_id=table2_id* The conditions needed to match

}

Reponse:
201 Created
400 Bad Request
401 Unauthorized access
412 Invalid arguments

Explanation:
This function deletes the record(row) of a specified table.

Assumption:
There are some pre-condition of this function. The first requirement is that the name must exist in the database. Moreover, The number of table should only be one

Limitation:
The advanced version has not yet been completed.

POST /table/data (insert one record into a single table)

Body:
{

name*	string *example: table1* The table name
columns*	string *example: table1.id* The row names
values*	string *example: 4440* User defined value of selected row

}

Reponse:
201 Created
400 Bad Request
401 Unauthorized access
412 Invalid arguments

Explanation:
This function adds a new record(row) into a specified table. There is a more advanced feature implemented for this function. For a table that has foreign keys to link to other tables, this function supports inserting into those linked tables by one command if there is no error.

Assumption:
There are some pre-condition of this function. The first requirement is that the name must exist in the database. Moreover, all the specified columns must be in that table, and there should not be any duplicate columns in the parameter. The length of the columns and the length of the values must match.

Limitation:
This function only supports inserting data into the tables directly linked to the specified tables, and this function only supports inserting one record at each time.

METADATA - Manage metadata

PUT /metadata (change the setting of columns in a table)

Body:
{

name* string
example: Table1
The name of the table to change the metadata

columns* string
example: col1,col2,col3,col3
A list of columns in the table to change the metadata(setting), and this parameter allow duplication as long as there are same number but different type of operations apply on that column

types* string
example: default, type, type, nullable

A list of operation to apply the column specified above, and this is an enum, the valid values are: default-to change default value of the column; type-to change the data type of the column; nullable-to specify if the column allow null value

values* string
example: something,int,varchar(100),yes
This is the list about what to do with each operation, for default, there is not requirement of value; for type operation, the valid values are int, float, double, decimal, date, string, char, and varchar; for nullable type, value yes, true and 1 are for allowing null value, and value no, false and 0 are for the opposite

}

Reponse:
200 OK
400 Table does not exist in the database
401 Unauthorized access

Explanation:
This function can change the metadata of a table, in which the settings of the columns in that table. These settings include default value, data type, and nullability. To change the default

value, the operation is 'default', and the valid value for this operation has no requirement as long as the database does not send errors. To change the data type, the operation is 'type', and the valid values are int(int), float, double, decimal(decimal), date(date), string, varchar(varchar), and char(char). To change the nullability, the operation is 'nullable', and the valid values are yes, true, and 1 for nullable, and no, false, and 0 for not nullable. As there are three possible operations on a single column, therefore, in the columns parameter, the same columns can appear as many as three times, and the requirement is that this column must have the same number of different operations, otherwise, there would be error.

Assumption:
Most importantly, the table must exist, and all the columns must be defined in the table. Moreover, the length of columns, operations, and values must be all equal. The arguments for the operation and value parameter must be a valid argument.

Limitation:
This function only supports very limited data types, namely, int, decimal, date, char(), and varchar().

GET /metadata (get the metadata)

Param:

table_name *	string *example: table1* Enter 'TABLE' to get a list of tables in the database; Enter 'VIEW' to get a list of views in the database; Enter an existing table name to get columns' information for that table.

Reponse:

200 OK
400 Table does not exist in the database
401 Unauthorized access

Explanation:
Get the metadata of the database or metadata of a certain table.

Assumption:
The table exists in the database.

METADATA/UNIQUEKEY - Manage unique key

DELETE /metadata/uniquekey (Delete unique keys(indexes) from a table)

Body:
{

name*	string *example: Table1* The name of table to modify
key_names*	string *example: key1,key2* A list of key names to drop from the table

}

Response:
201 Created
400 Bad Request
401 Unauthorized access
412 Invalid arguments

Explanation:
This function drops indexes from a specified table.

Assumption:
The table must exist, and the key names must be defined in that table.

POST /metadata/uniquekey (add a new index to the table)

Body:
{

name* string
example: Table1
The name of table to modify

keys* string
example: col1,(col3,col2)
A list of columns to add unique keys(indexes), and comma is used to separate each column, and parentheses is used to group composite key

key_names* string
example: index1,index2
A list of names for the new keys

}

Reponse:
200 Created
400 Bad Request
401 Unauthorized access
412 Invalid arguments

Explanation:
This function adds new unique keys(indexes) to the specified table. The key field is a list of columns to add a new index, and the field key_name is a list of the names to these new indexes.

Assumption:
The table must exist; the columns must be defined; the key names must be new; the length of keys and key_names must match. Any one of the requirements fails will cause the fail of the whole function. For composite keys, all elements in a parentheses are counted for one in terms of matching the length between the keys parameter and the key_names parameter.

GET /metadata/uniquekey/{table_name} (Get a list of unique keys reference of a table)

Params:

table_name*	string *example: table1* An existing table name.

Reponse:

201 Created
400 Bad Request
401 Unauthorized access
412 Invalid arguments

Explanation:
This function returns a list of unique keys(indexes) that is defined in the specified table. If there is no index on that table, then it returns null value.

Assumption:
The table must exist in the database.

METADATA/FOREIGNKEY - Manage foreign key

DELETE /metadata/foreignkey (Delete foreign keys(references) from a table)

Body:
{

name* string
example: Table1
The name of table to modify

key_names* string
example: key1,key2
A list of key names to drop from the table

}

Response:
201 Created
400 Bad Request
401 Unauthorized access
412 Invalid arguments

Explanation:
This function drops references from a specified table.

Assumption:
The table must exist, and the key names must be defined in that table.

Limitation:
The foreign key can only be deleted from the table which references to others but not the table which is referenced by others.

POST /metadata/foreignkey (Add a new index to the table)

Body:
{

name* string
example: Table1
The name of table to modify

keys* string
example: col1,col2
A list of columns to add foreign keys(references), and comma is used to separate each column

targets* string
example: Table2.col1,Table3.col2

A list of columns that the keys reference to, and the format is 'tablename.columnname'

key_names* string
example: reference1,reference2
A list of names for the new keys

}

Reponse:
201 Created
400 Bad Request
401 Unauthorized access
412 Invalid arguments

Explanation:
This function adds new foreign keys(references) to the specified table. The key field is a list of columns to add a new ireference, and the field key_name is a list of the names to these new indexes. The field targets is the list of columns that the keys reference to.

Assumption:
The table must exist; the columns must be defined; the key names must be new; the length of keys, targets and key_names must match; the target value must in the correct format and both the table name and the column name must be defined. Any one of the requirements fails will cause the fail of the whole function.

Limitation:
This function does not support add composite freign key(reference) now.

GET /metadata/foreignkey/{table_name} (Get a list of foreign keys reference of a table)

Params:

table_name*	string *example: table1* An existing table name.

Reponse:

201 Created
400 Bad Request
401 Unauthorized access
412 Invalid arguments

Explanation:
This function returns a list of foreign keys(references) that is defined in the specified table. If there is no reference on that table, then it returns null value.

Assumption:
The table must exist in the database.

Limitation:
This function can only return the foreign keys that this table has to reference to another table, but not the foreign keys that the other tables have to reference to this table.

UNION

POST /union (union two existing tables from the database)

Body:
{

table_name_A*	string *example: Table1* An existing table name.
columns_A*	string *example: col1,col2* Specify the column to retrieve from table A and separate each column name by comma. Select ALL if leave it blank
table_name_B*	string *example: Table2* Specify the column to retrieve from table B and separate each column name by comma. Select ALL if leave it blank
columns_B*	string *example: col1,col2* Specify the column to retrieve from table B and separate each column name by comma. Select ALL if leave it blank
returned_view_name*	string *example: view_x* Name the view if you want to save the result as a view.

}

Reponse:
201 Created
400 Bad Request
401 Unauthorized access
412 Invalid arguments

Explanation:
Check whether input tables and columns are valid and then union selected columns.

Assumption:
If leave ‘columns_A’ and ‘columns_B’ blank, it will automatically select ALL from two tables and union. The number of columns in these two fields must match.

GROUPBY

POST /groupby (apply group by function to a table and create a view)

Body:
{

name*	string *example: Table1* The name of table to modify
functions	string *example:* A list of MySQL predefined functions with parameters, each function shall in form of 'function(param)' and should be separated by comma from other functions
rename	string *example: count,maximum* A list of name of the result from the functions, and each name is corresponding to one function, and each rename is separated by comma
group_by*	string *example: col2* This field specifies according which data the table should be grouped
view_name*	string *example: grouped1* This the the specified name for the view created by this function

}

Reponse:
201 Created
400 Bad Request
401 Unauthorized access
412 Invalid arguments

Explanation:
This function applies group by function to a table and creates a temporary view to same the result for future usage. This function also supports calling the MySQL predefined functions on certain columns.

Assumption: The table must exist, the view must not exist before this function, the length of functions must match the length of renames, all the functions must be defined and used correctly.

Limitation:
This function and the Union function, the Join function are created based on this concept: a complex and long MySQL query needs to be decomposed to make it easier for humans to understand. Therefore we create these three functions to create a stage view for each, and the user can do more queries on these temporary views to accomplish the complex query. This mechanism makes the query easy to understand, but it requires more simple queries to accomplish the same goal.

JOIN

POST /join (Apply join function to selected tables and create a view)

Body:
{

tables*	string *example: table1* The table names
columns*	string *example: table1.id* The column names
renames*	string *example: table1_id* User defined name of columns
joinType*	string *example: inner* Jointype: inner, partial, full
match*	string *example: table1_id=table2_id* The conditions needed to match
returned_view_name*	string *example: view_name* User defined name of view

}

Reponse:
201 Created
400 Bad Request
401 Unauthorized access
412 Invalid arguments

Explanation:
This function applies a join function to a table and creates a temporary view to same the result for future usage.

Assumption: The table must exist, the view must not exist before this function, the length of functions must match the length of renames, all the functions must be defined and used correctly.

Limitation:
This function and the Union function, the Groupby function are created based on this concept: a complex and long MySQL query needs to be decomposed to make it easier for humans to understand. Therefore we create these three functions to create a stage view for each, and the user can do more queries on these temporary views to accomplish the complex query. This mechanism makes the query easy to understand, but it requires more simple queries to accomplish the same goal.

FILTER

POST /filter (create a view for filter a table)

Body:
{

name* string
example: table1
The name of table to modify

columns* string
example: col1,col2,col2
A list of columns that need to be tested in the conditions, and duplication is allowed. The columns should be separated by comma

operators* string
example: LIKE,<,~IN
A list of operators to apply to the columns, and the valid operator includes =, >, <, >=, <=, !=, BETWEEN, LIKE, IN and negation of them. To test the negation of the condition, please add ~ symbol before the operator. The operators should be separated by comma

conditions* string
example: %ello,10,(3,5,11)
A list of conditions that works with each operator. For the condition of IN and BETWEEN operator should be grouped by parentheses

type* string
example: AND
Define how to conjunct each conditions, valid values are AND, OR, XOR

view_name* string
example: filter1
This the the specified name for the view created by this function

}

Reponse:
201 Created
400 Bad Request
401 Unauthorized access
412 Invalid arguments

Explanation:

This function creates view according to the conditions listed by user, in which the conditions are corresponding to the where clause in MySQL.
Assumption:
For each boolean evaluation, this function requires three parts: column, the value on the left of the logic operator; operator, the logic operator; condition, the value on the right of the logic operator.
Limitation:
This function only supports linear conjunction between boolean evaluations, and for each call, it only allows one kind of conjunction, and, or, xor.

UPLOAD

POST /upload (upload a csv file to the database)

Body {

name string
example: Table1
An existing table name

csv* string
example:
http://samplecsvs.s3.amazonaws.com/Sacramentorealestatetransactions.csv
Url of an csv file

}

Response:
201 Created
400 Failed to download the csv file
401 Unauthorized access

Explanation:
This function processes the uploaded csv by creating a table with the table name and inserting each row to the table. The column name is the header of the csv. If the table name is specified, the data will be added to an existing table. Otherwise, it will create a new table.

Assumption:
The table name must not exist in the database and the file url must be valid.

Limitation:
The file url must end with .csv format. This function does not create any key