# 

[Appup](#_5u4seijstmnm)

[DB team](#_hswpbdf67bod)

[Postman collection](#_k5hxlu6to4g0)

[CRUD Operations](#_mj5pylvvgmor)

[Insert multiple entities (POST)](#_gcp4a27vhfb)

[Update single or multiple entities for a specific column (PUT)](#_6p72d7l52kba)

[Delete multiple entities by id (DELETE)](#_jtbfbqqau2ki)

[Retrieve an entity by id (GET)](#_aluun3lnmq64)

[Retrieve entity(ies) (GET)](#_mx7g3urz4vrd)

[Read (Misc)](#_vbtoxp94to1r)

[BULK UPDATE](#_o3fg23xal8lm)

[Upsert](#_kjci0vrw7kqu)

[Tags](#_l0pxtottrsn8)

[Associate a tag to an one or more entities (POST)](#_e3aqnwaj7fxs)

[Remove a tag for one or multiple entities (DELETE)](#_dit6mg89gond)

[Retrieve all entities with a given tag(GET)](#_li5g6uhod8xb)

[TAG Management](#_bg64smdmtqw9)

[Filters (POST)](#_skpq8uff536s)

[Exec-SP](#_bseh54l5cm0x)

[Import/Export - Work In progress](#_qg8uw4nuchaq)

[Join](#_gzmal52gl4wf)

[Inner Join](#_tkvc1qdlbjz6)

[Left Join](#_6q58r24ixywq)

[Self Join](#_6i1gsc82vog8)

[Right Join](#_9qsdrhr61hs0)

[Custom Fields](#_rwmzbrf53268)

[Reports - Not yet started](#_584eu7qos632)

[Default Values](#_xyigxnjqq0ed)

[New account sign up](#_mhnzuxi2jyni)

[Invite new user](#_gn1l8qaaoyas)

[Activity](#_4qhvi4474ch2)

[Versioning](#_16v6gve4krnz)

[Security - JWT](#_7qx58qh5n47l)

[Insert table with JWT](#_8ll0927a3g6)

[The browser makes a POST request to the server that contains the domain information. The server responds with a cookie, which is set on the user’s browser, and includes a session ID to identify the domain.](#_c2esca517tzt)

[JWT Filter](#_np0haqgbfe50)

[DESCRIBE](#_g56iajllm9pk)

[Search in multiple entities](#_v9s20oa1yo9i)

[Conflict Check](#_7dx81fhigra5)

[Checks](#_y0c5ghqydmqk)

[JSON Search](#_kg3g416txsl5)

[Merge Duplicates](#_waxhqi2stafa)

[IP LOCATIONS](#_zbsxr5d2g51r)

[Email Reporting](#_c1wxu7vk47ep)

[Db Form](#_5383296gxsdp)

[Query Builder From UI](#_t4pxesgqgstf)

[Access Control Lists (ACL’s)](#_sjdg2yfyqg4x)

[Custom Queries](#_ma9sqttwv3ol)

[Regular Expressions](#_vvegqrfwy8hm)

# Appup

With the proper setup of workflows, Appup can do most of the heavy lifting of the backend without writing much of the code.

# DB team

DB team (around 10 members) has been created to help all engineering teams with their database requirements.

Expectations:

* Creation of schema for new apps based on the functionality document
* Setup of the database in Aurora
* Copying of Appup DB app in new project (which has all generics) as /v1/ or /\_db/
* Setup of JWT Filters and its interoperability in DB (sets domain\_id)
* Backup of database records
* Fine-tuning of database queries
* Database tuning
* Creation of default data for each new users signing up based on the functionality
* Weekly reports of each database (snapshot)
* Postman collection
* Versioning

# Postman collection

V1 https://drive.google.com/open?id=1igYCZ2CVAm35-MmxSwEkUifiJR4\_ssKK

V1.1 https://drive.google.com/open?id=14r7KDi2eIxySQrk0gA1JPLKJkaxiv2Y4

# CRUD Operations

## Insert single or multiple entities (POST)

Appup will insert multiple rows or bulk of records into the specified table.

Trigger Expression:/v1/{table\_name}

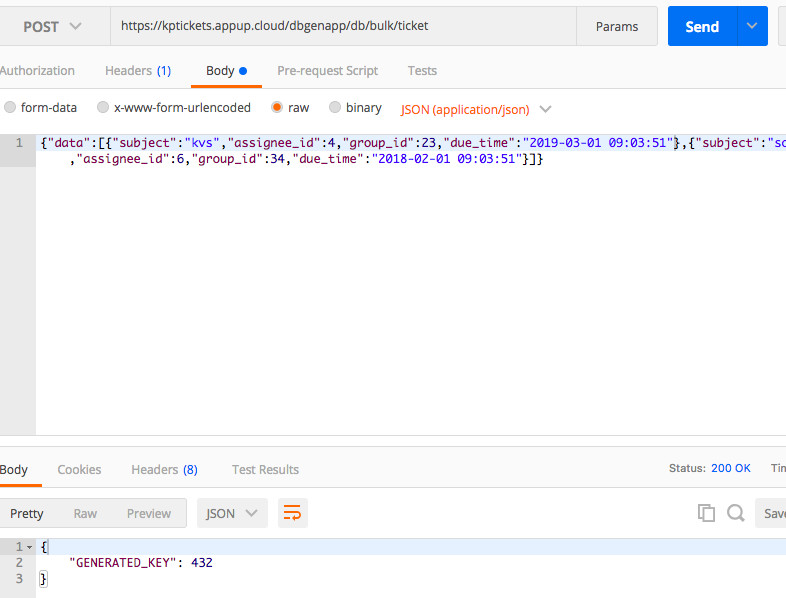
Method:POST

Example:

Let’s take an example where required to insert multiple tickets into **ticket** table. The values will given as part of request body. Find below postman notation for ready reference.

The input data would be sent with content-type as **application/json** as shown in below postman snapshot. The data is expected as JSON Array.

Rest Call: https://dbapp.500apps.com/v1/ticket

Returns

The response code is 200 OK. Currently the generated key of just one row is returned back by Appup. This needs to be corrected.

## Update single or multiple entities for a specific column (PUT)

Appup will update multiple rows in the specified table based on id.

Trigger Expression: /v1/{table\_name}

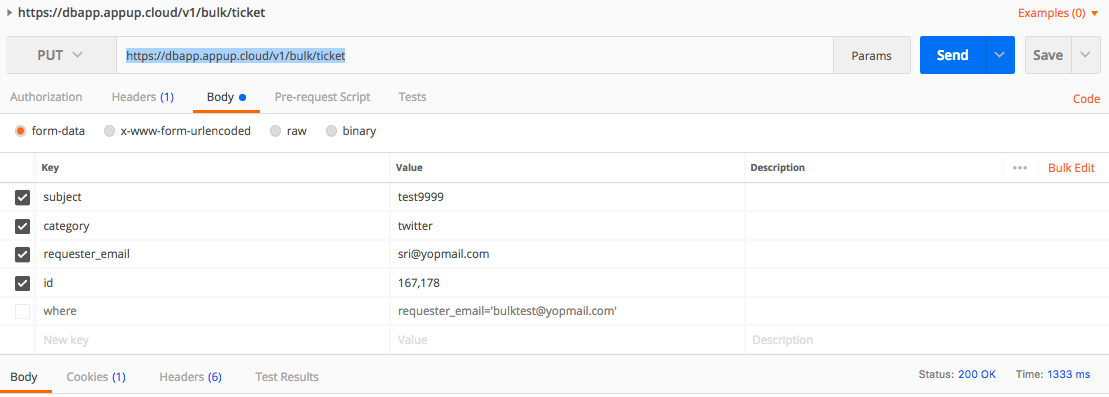
Method:PUT

Example:

Let’s take use case of update multiple tickets in the ticket table for field subject, email, and category. We will pass the data as part of body.

The id and the values can be passed as comma separated. The field to be updated is mandatory too.

Rest Call: “PUT”: “https://dbapp.appup.cloud/v1/ticket”



Returns

The number of rows updated would be returned

## Delete multiple entities by id (DELETE)

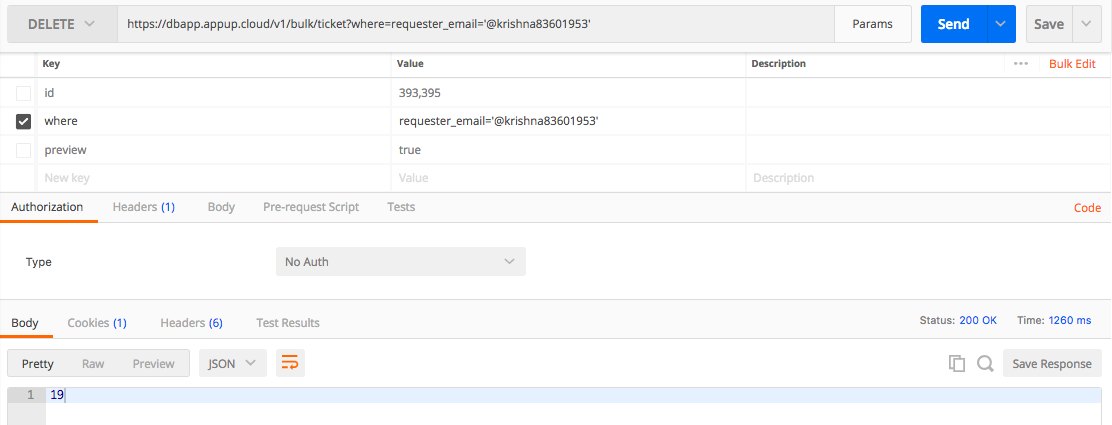
Delete is a Appup plugin which can be used to delete data in bulk for the selected records.

Trigger Expression:/v1/{table\_name}

Method:DELETE

The id(s) has to be passed as part of params or they can be passed the condition in where.

Rest Call: https://dbapp.appup.cloud/v1/ticket?where=requester\_email='@krishna83601953'



Returns

The number of rows deleted shall be returned in response

{

## Retrieve an entity by id (GET)

Appup will retrieve the data from the specified table for the given id.

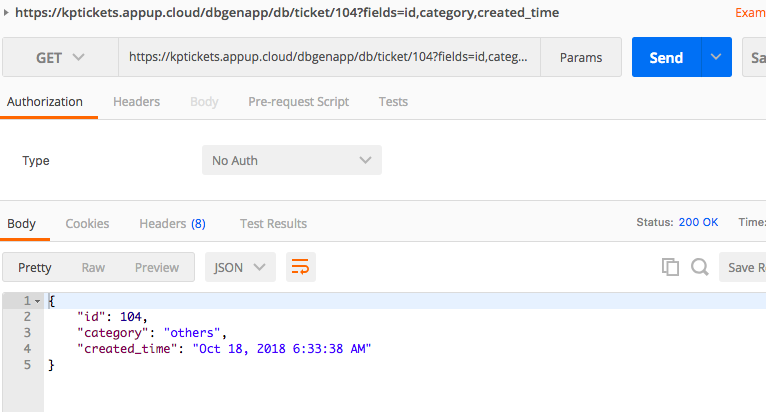
Trigger expression /v1/{table\_name}/{id}/?fields=category,id,subject

Method: GET

Example:

Let’s take an example where we want to get a single **ticket**. Below notation we can use in Postman. The fields are optional when not specified all the values for the specified ticket id would be retrieved

Rest Call: https://kptickets.appup.cloud/v1/ticket

* 

Returns

A 200 response code would be given back with below data.

{

"id": 178,

"subject": "test",

"created\_by": 4

}

## Retrieve entity(ies) (GET)

Appup will retrieve the data from the specified table. It also supports various operations like to filter the data using where clause, ordering the data in ascending/descending, pagination, limit the no of rows returned.

Trigger expression: /v1/{table\_name}?fields=&where=&order\_by=&order\_by\_type=&limit=&offset=

Method GET

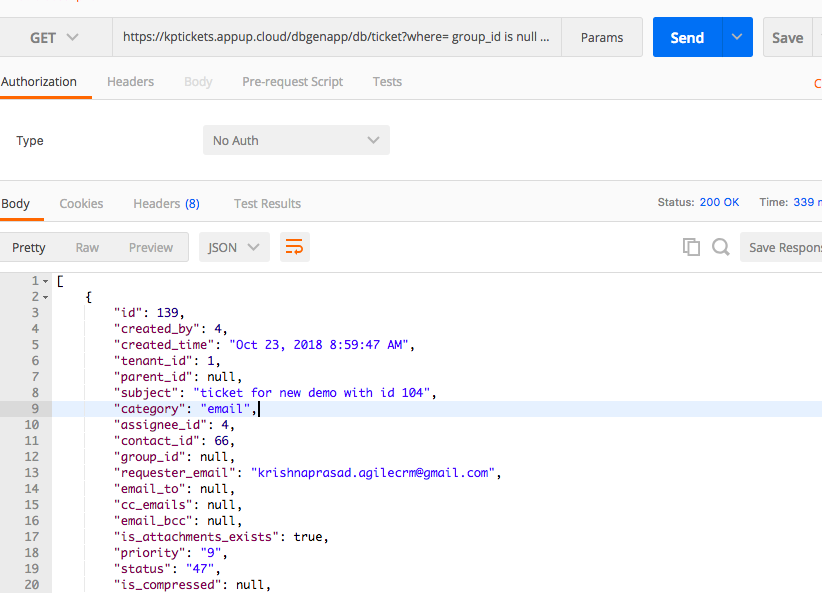
Example:

Let’s take an example where we want to get the data **ticket** table for **all fields**. The params has to be passed as parameters. The postman example is given here as reference.

Available Params. All are optional

* Fields - specify the field names
* Group by - For grouping columns
* Where - For filtering
* Order By - Sorting column
* Order Type - For sorting ascending or descending
* Limit - Limit of records required
* Offset - to specify which row to start from retrieving data. This will be used for pagination

Rest call: https://kptickets.appup.cloud/dbgenapp/db/ticket?where= group\_id is null and assignee\_id is not null&order\_by=assignee\_id&order by type=asc



Returns

A JSON array is returned that satisfies the search criteria with all the fields as mentioned.

## Read (Misc)

Appup will retrieve the aggregate data from a specific table. Aggregate functions supported are sum, count, max, min, avg. The input has to be passed as part of path and params.

Trigger expression:

/v1/aggregate/{table\_name}/{func name}?aggregate\_field=&fields=&group\_by=&order\_by=

Method: GET

Different function names supported are

Sum

Count

Max

Min

Avg

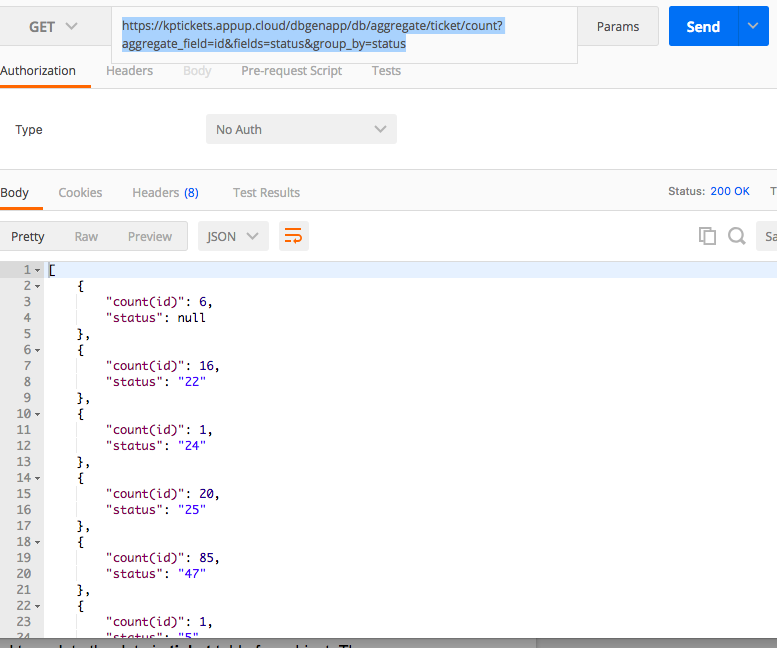
Example:

Let’s take an example where required to get the count of tickets from **ticket** table based on ticket status.

Rest Call Notation: [https://kptickets.appup.cloud/dbgenapp/v1/aggregate/ticket/count?aggregate\_field=id&fields=status&group\_by=status](https://kptickets.appup.cloud/dbgenapp/db/aggregate/ticket/count?aggregate_field=id&fields=status&group_by=status)

https://kptickets.appup.cloud/v1/ticket

The postman example is given here as reference.



Returns

A JSON array is returned that satisfies the search criteria with all the fields as mentioned.

# **BULK UPDATE**

Updation of different rows with different conditions is bulk update. We can update any number of rows at a time with multiple conditions. As in the normal update which we have seen previously,here number of columns to be updated need not be same or equal in column count.

Example : Here updation should be done like :

when id={{id}} then column={{value}},

when id={{id}} then column1={{value1}},column2={{value2}}

In brief :

when id = 1 then update name = ‘raghu’

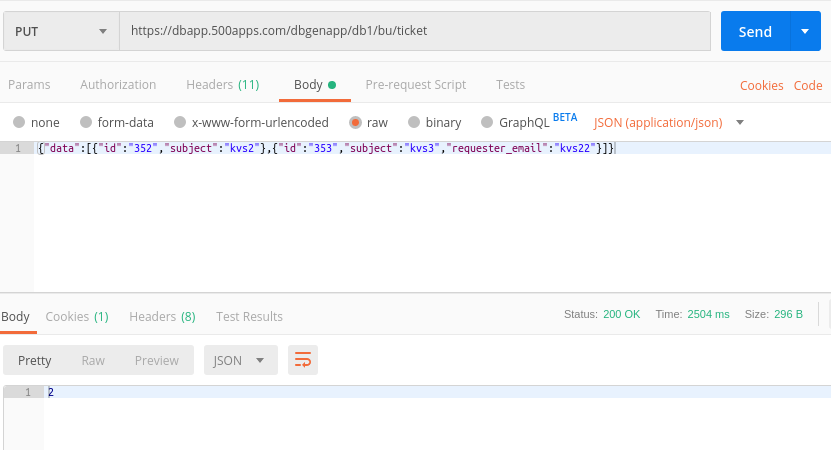
when id = 2 then update subject=’test1’ and category=’twitter’

Let us see the example through postman

**Trigger Expression** : /multi/{table\_name}

**Method** : PUT

**Rest call** : [https://dbapp.500apps.com/v2/multi/ticket](https://dbapp.500apps.com/dbgenapp/db1/v2/ticket)



# Upsert

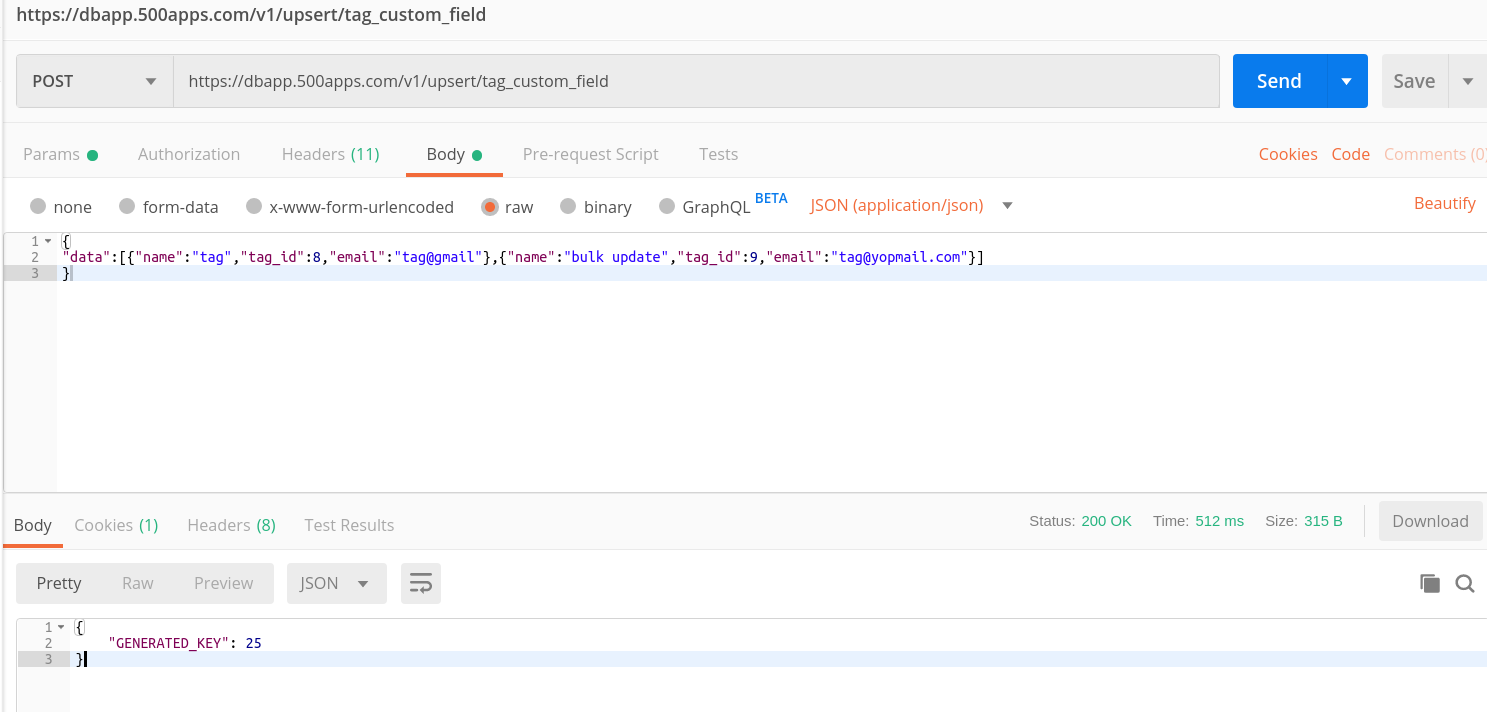
Update if exists else insert.Need to specify unique index for columns in table if their is condition.

Let us take an example of tag\_custom\_field table,for this table there is a condition that it should take only one combination of tag\_id and created\_by.so we need to specify unique index for tag\_id and created\_by in table.Parameters will be sent as body.

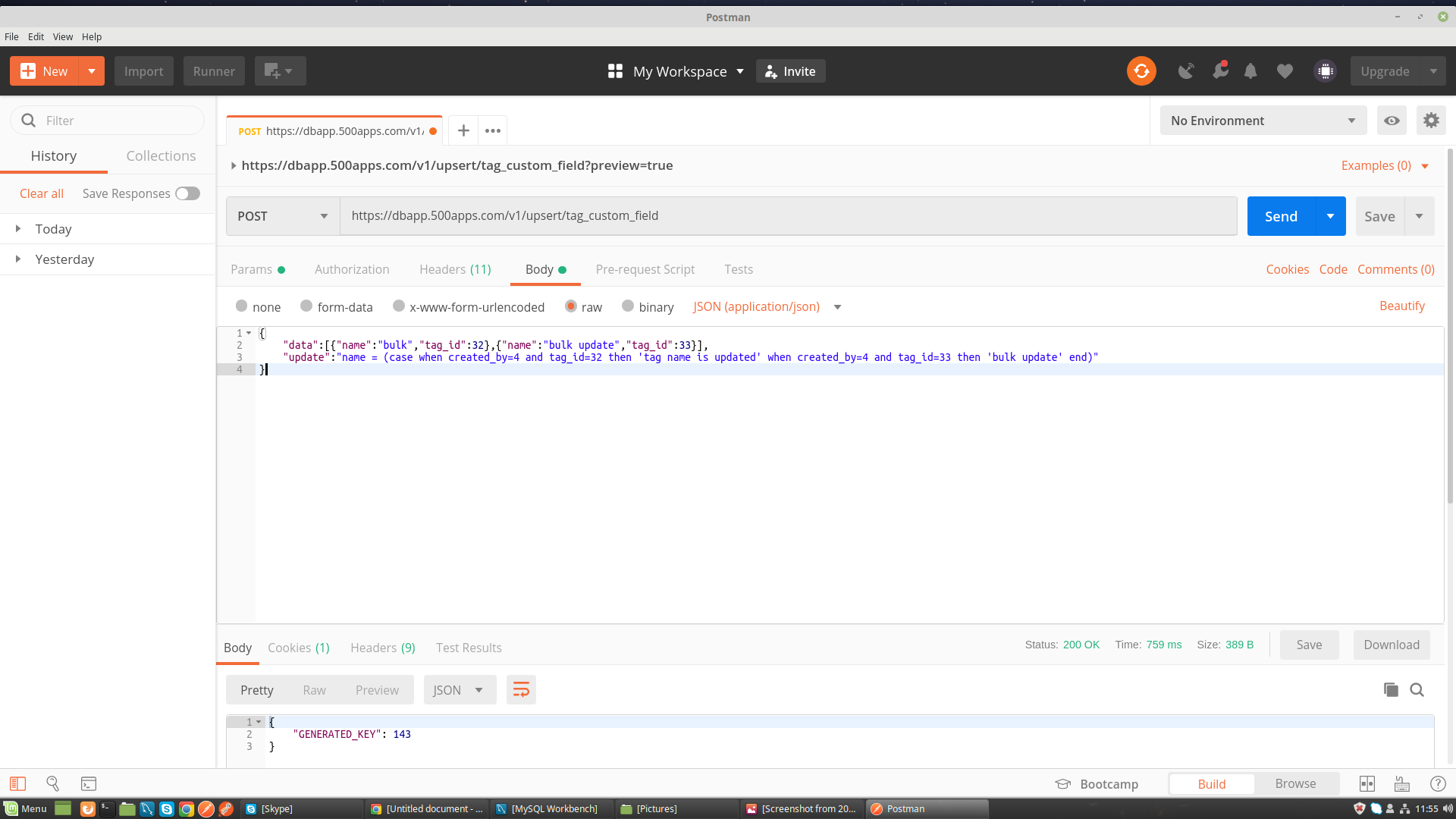
Trigger expression : /upsert/{table\_name}

Method : POST

Rest call : “POST” :”<https://dbapp.500apps.com/v1/upsert/tag_custom_field>”



Returns : If insertion is done,new generated key will display. When updation is done updated key will display.



Returns : if insertion is done,new generated key will display. When updation is done updated key will display.

# Tags

The CRUD section may be referred to create, retrieve, update or delete a tag. The table name for “tag” may be mentioned. However there are postman collections given for reference.

We will discuss more about how a tag may be associated/removed to an entity say ticket

## Associate a tag to an one or more entities (POST)

Here the assumption: the table is like contact\_tag, ticket\_tag, article\_tag exists. We concatenate the entity with suffix \_tag. The entity id(s) are sent as part of the body in JSON array.

Trigger Expression: /v1/tags/{entity\_id}/{table\_name}

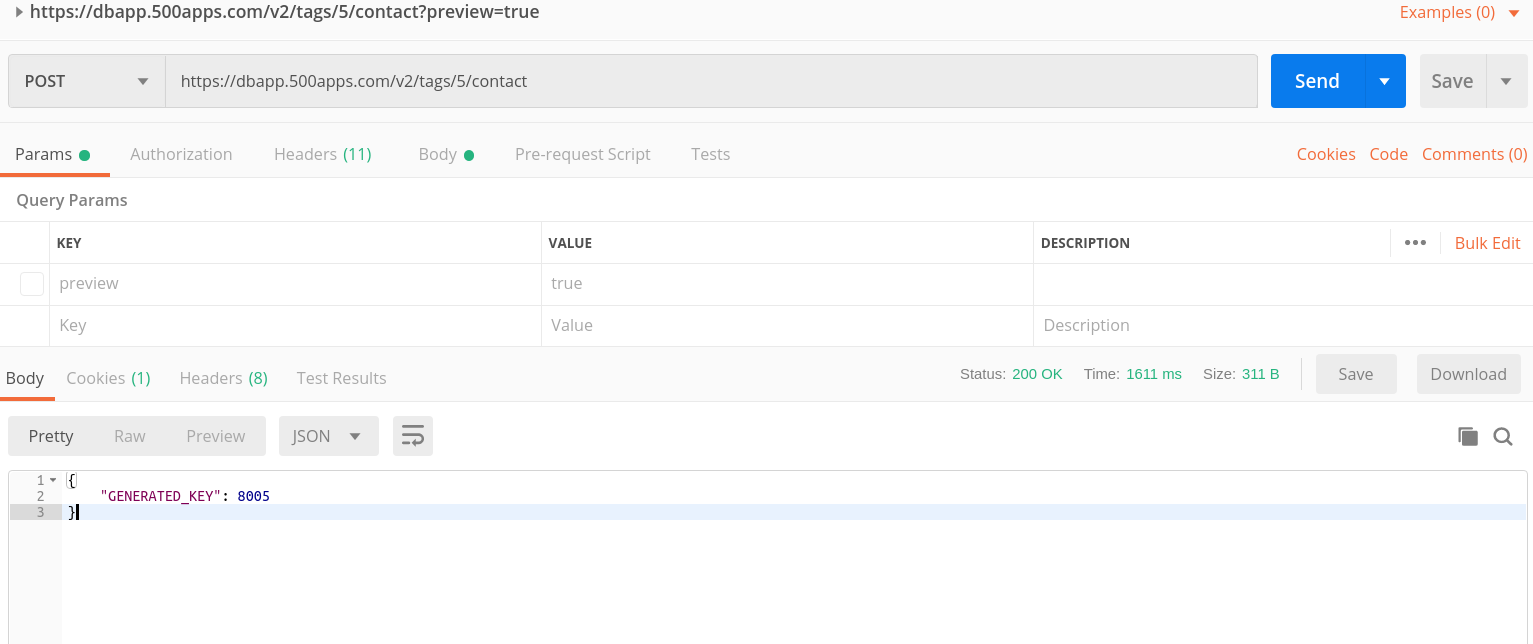
Method: POST

Content-Type: application/json

Example:

In our example the contact\_id would be passed as part of JSON Array as shown in the POSTMAN example

Rest call::<https://dbapp.500apps.com/v2/tags/5/contact>



Returns

A 200 OK response code with the generated key would be returned. Here Appup returns only one generated key.

## Remove a tag for one or multiple entities (DELETE)

This api shall be used to delete a tag associated with one or more entities.

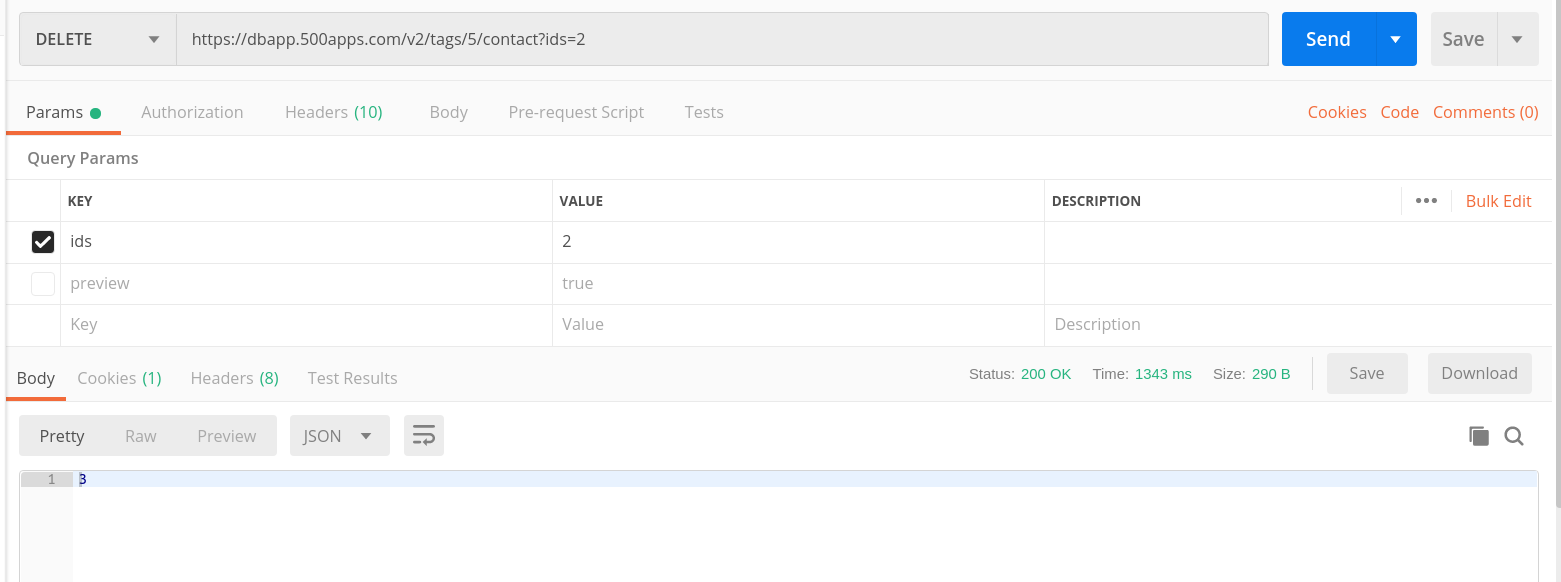
Trigger Expression: /v1/tags/{entity\_id}/{table\_name}/?ids=

Method: DELETE

Example:

The entities ids has to be specified in query params with optional comma separator when multiple ids

REST Call: https://dbapp.500apps.com/v2/tags/5/contact?ids=2



Returns

A 200 OK response code with the number of rows deleted would be returned.

## Retrieve all entities with a given tag(GET)

This api shall be used to retrieve all entities associated with the given tag name. Fields needed from the entity may be mentioned as part of the query params.

The assumption is there is a table called tag and tablename\_tag like contact\_tag or ticket\_tag etc

Trigger expression: /v1/tags/{table\_name}/fields=&tagname=&order\_by=&order\_by\_type.

Method: GET

Example:

Let’s take an example say to retrieve all contacts that has a tag “messi”

Rest call :

https://dbapp.500apps.com/v2/tags/contact?tag\_name=Team&fields=first\_name,last\_name,email&limit=6



Returns

A 200 OK response code would be sent along with JSON array of all matched contacts.

## TAG Management

# Filters (POST)

This rest api shall be used to specify various filter criteria and retrieve the data from several tables.

Trigger Expression: /db/filters

Method: POST

Support Join types

JOIN

LEFT JOIN

RIGHT JOIN

Supported Data Types

String

Number

Date

Boolean

Supported Operators

EQUALS - User has to pass EQ in the attribute

NOT EQUALS - User has to pass NE

LIKE - User has to pass LIKE

NOT LIKE User has to pass NOTLIKE

IN user has to pass IN

NOT IN user has to pass NOTIN

IS NULL User has to pass NULL

IS NOT NULL User has to pass NOTNULL

BETWEEN User has to pass BW

NOT BETWEEN User has to pass NBW

STARTS WITH User has to pass STARTSWITH

NOT STARTS WITH, User has to pass NOTSTARTSWITH

Ends with User has to pass ENDS

Not Ends with User has to passNOTENDS

To check empty data - EMPTY

To check non empty data-NOTEMPTY

> - User has to pass GT

< - User has to pass LT

>= User has to pass GTE

<= User has to pass LTE

The input data has to be sent as part of the request body as JSON format. A sample is given below. Any attribute that is not needed for a particular operation should not be included.

request={

"body":{"data":{

"fields": "t.id,t.name,t.status,tn.html\_text",

"tables":[{"ticket":”t”},{"ticket\_note":”tn”}],

"join":[{

"join\_type":"left join",

"table1":"t",

"join\_column1":"id",

"table2":"tn",

"join\_column2":"ticket\_id"

}],

"filters": [

{

"filter\_condition" :"",

"table\_name" : "t",

"field\_name" : "status",

"operator" : "GT",

"value1" :"14",

"value2" :" "

},

{

"filter\_condition" :"and",

"table\_name" : "t",

"field\_name" : "priority",

"operator" : "NE",

"value1" :"14",

"value2" :" "

}

],

"order\_by":"t.id",

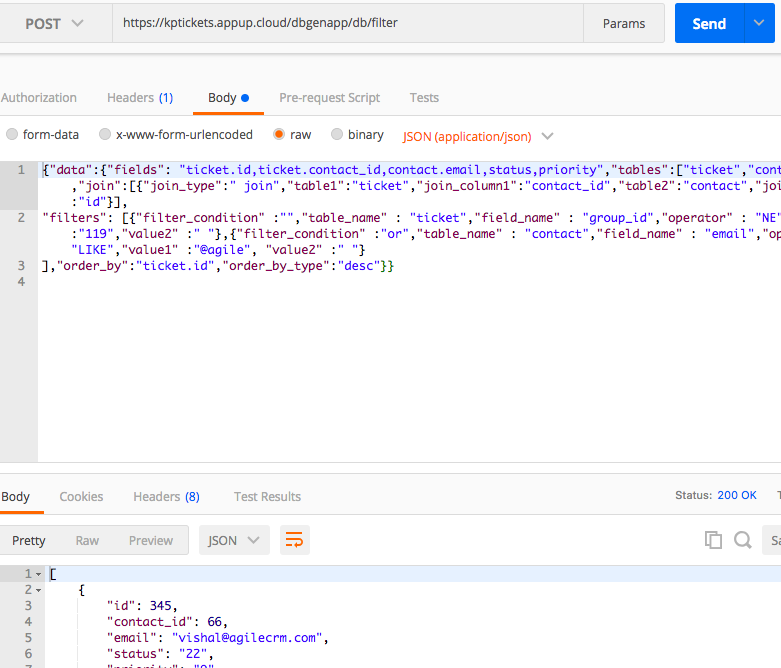
"order\_by\_type":"asc"

}

}

}

Rest Call: https://kptickets.appup.cloud/dbgenapp/db/filter



Returns

A 200 OK response code would be returned along with the JSON array containing the filtered data.

# Exec-SP

This workflow will be used for execute a procedure in Appup. Putting database-intensive operations into stored procedures lets you define an API for your database application. You can reuse this API across multiple applications. This technique avoids duplicating database code, saving time and effort when you make updates due to schema changes, tune the performance of queries, or add new database operations for logging, security, and so on.

Supported Operations Store procedures:

* Without In and Out Parameters
* With In Parameter

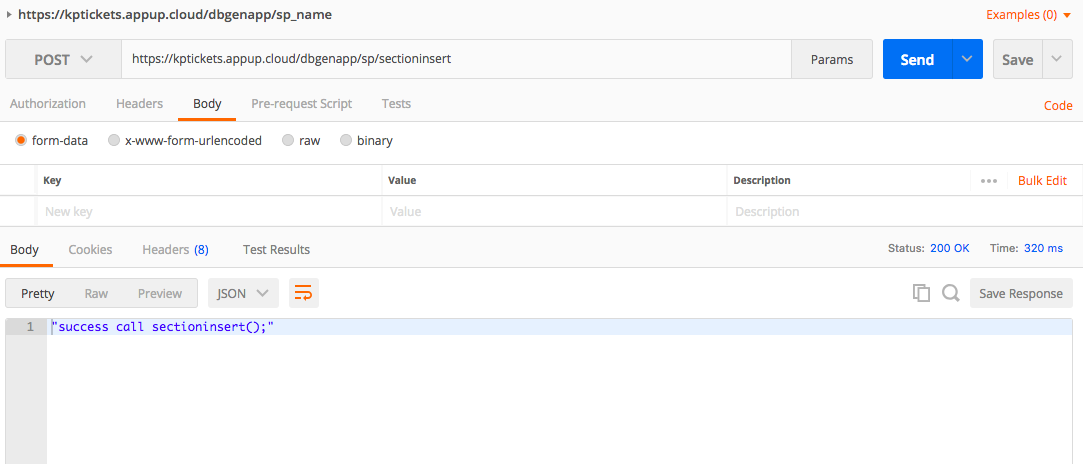
**Without In and Out Parameters**

Let us take example of where we need to do set of transactions like insert multiple tables at one time we will use this stored procedure appup. If parameters is required in stored procedure will be provided in param values otherwise we will leave blank.

Trigger expression is :/v1/sp/{sp\_name}

Method: POST

Rest Call Notation: “POST”:”https://kptickets.appup.cloud/dbgenapp/sp/sectioninsert”



Returns

A 200 OK response code would be returned along with success call stored procedure name.

**With In parameters**

Trigger expression is :/v1/sp/{sp\_name}

Method: POST

Rest Call:”https://kptickets.appup.cloud/v1/sp/insertt?parameters=3,'tag test',24

# 

Returns

A 200 OK response code would be returned along with success call stored procedure name.

# Import/Export - Work In progress

**Import**

This workflow is used for import data from csv file.

Let’s take an example where required to import the data into **staging** table with csv file.

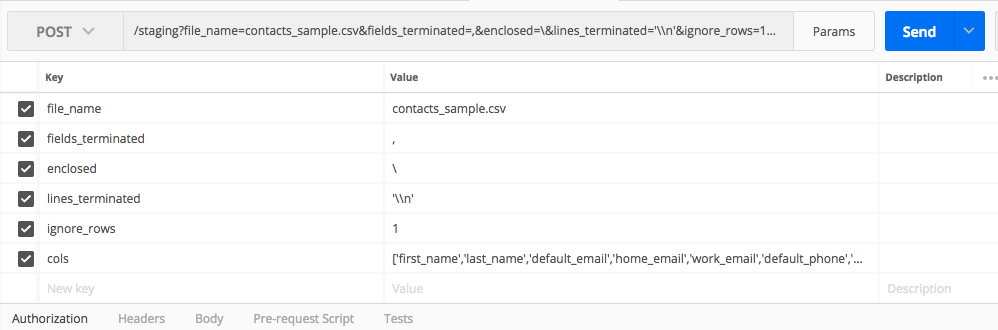
The values has to be passed by params.

Below Params are require:

* File Name
* Filed\_terminated
* Enclosed
* Lines\_terminated
* Ignore\_rows
* Columns names

/import/{table-name} (POST)

Rest call : “POST”:”/staging?file\_name=contacts\_sample.csv&fields\_terminated=,&enclosed=\&lines\_terminated='\\n'&ignore\_rows=1&cols=['first\_name','last\_name','default\_email','home\_email','work\_email','default\_phone','work\_phone','home\_phone','address','city','state','country','zip\_code','default\_website','website','skype\_name','twitter\_url','linkedin\_url','facebook\_url','custom1','custom2','custom3','custom4','tags','@manual\_tags','@tenant\_id']”



# 

**Export:**

Export is doing by Stored procedure.

# Join

## Inner Join

This appup will be used for joining multiple tables and get records..

The Inner Join selects records that have matching values in both tables.

Let us take example of where ticket table joining with domain table getting domain name.

The input data has to be sent as part of the request body as JSON format. Any attribute that is not needed for a particular operation should not be included.

Trigger Expression:/v1/join/

Method:”POST”

Content-Type: application/json

A sample is given below:

request={

"body":{"data":

{"fields": "du.name domain\_name,t.id,t.subject",

"tables":[

{"domain\_user":"du"},

{"ticket":"t"}

],

"join":[

{

"join\_type":"join",

"table1":"du",

"join\_column1":"id",

"table2":"t",

"join\_column2":"created\_by"

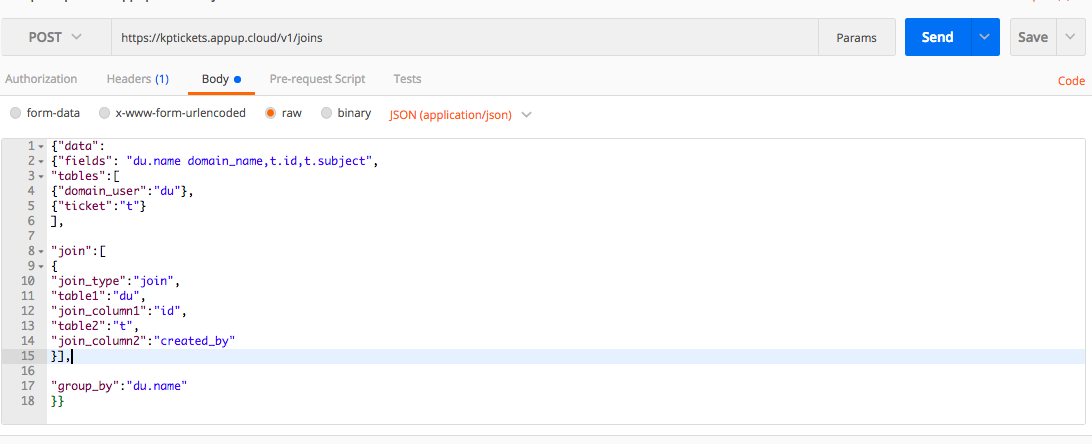
}],

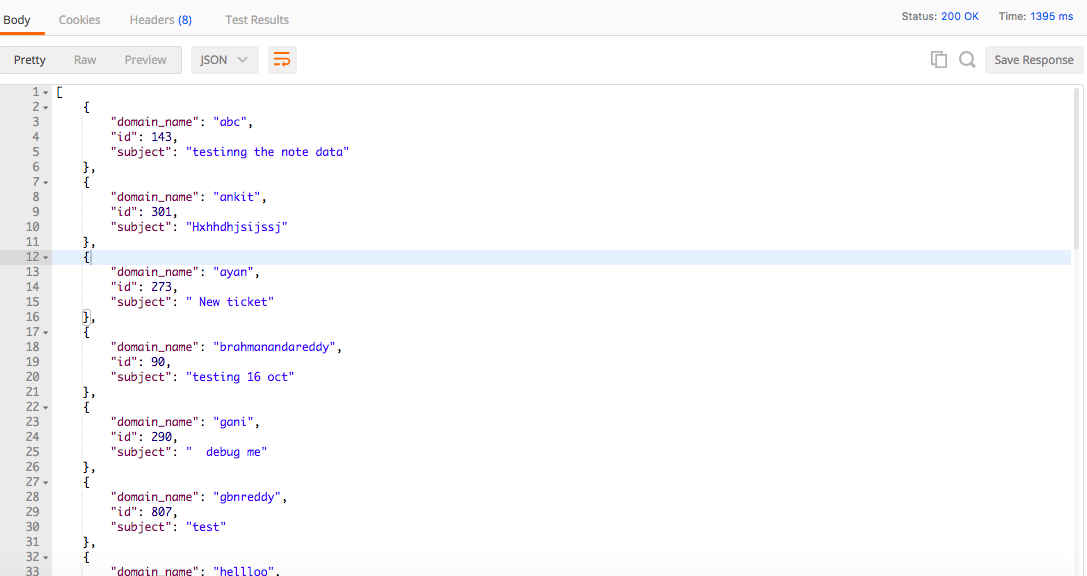
"group\_by":"du.name"

}}

}

RestCall:”POST”: “https://kptickets.appup.cloud/v1/joins”

Returns:



## Left Join

This workflow returns all records from the left table (table1), and the matched records from the right table (table2). The result is NULL from the right side, if there is no match.

Let us take example for contact and ticket tables joining(left), some tickets does not have matched records with contact, so its displays null or blank values.

The input data has to be sent as part of the request body as JSON format. Any attribute that is not needed for a particular operation should not be included.

Trigger Expression:/v1/join/

Method:”POST”

Content-Type: application/json

A sample is given below:

request={

"body":{"data":

{"data":

{"fields": "t.id,c.first\_name,c.company",

"tables":[

{"contact":"c"},

{"ticket":"t"}

],

"join":[

{

"join\_type":"left join",

"table1":"c",

"join\_column1":"id",

"table2":"t",

"join\_column2":"contact\_id"

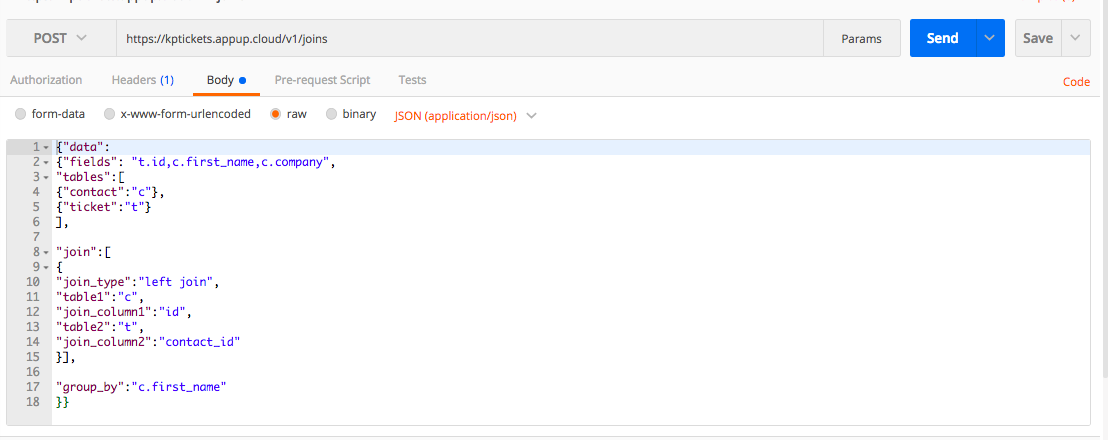
}],

"group\_by":"c.first\_name"

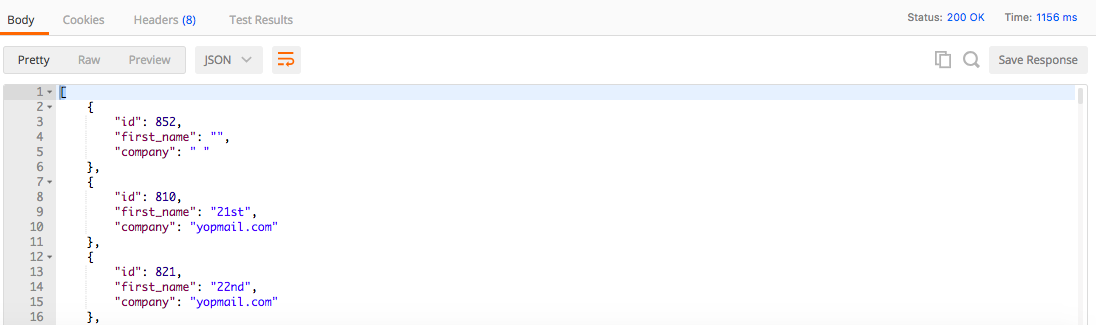
}}

}

RestCall:”POST”: “https://kptickets.appup.cloud/v1/joins”



Returns:



## Self Join

A self JOIN is a regular join, but the table is joined with itself. Self-joins are used to compare values in a column with other values in the same column in the same table.

Let us take example for ticket table, parent\_id are located in ticket table with id.

The input data has to be sent as part of the request body as JSON format. Any attribute that is not needed for a particular operation should not be included.

Trigger Expression:/v1/join/

Method:”POST”

Content-Type: application/json

A sample is given below:

request={

"Body": {"data":

{"fields": "t1.id,t1.subject, t2.id,t2.subject",

"tables":[

{"ticket":"t1"},

{"ticket":"t2"}

],

"join":[

{

"join\_type":"join",

"table1":"t1",

"join\_column1":"id",

"table2":"t2",

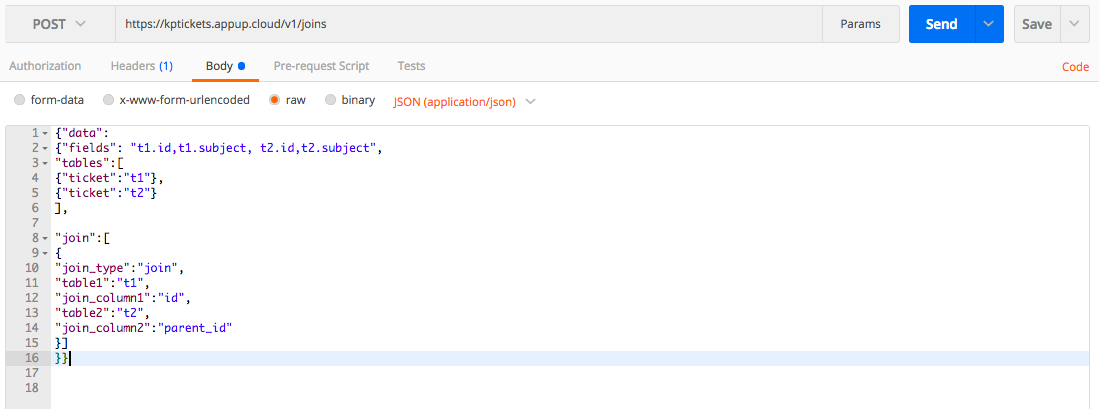
"join\_column2":"parent\_id"

}]

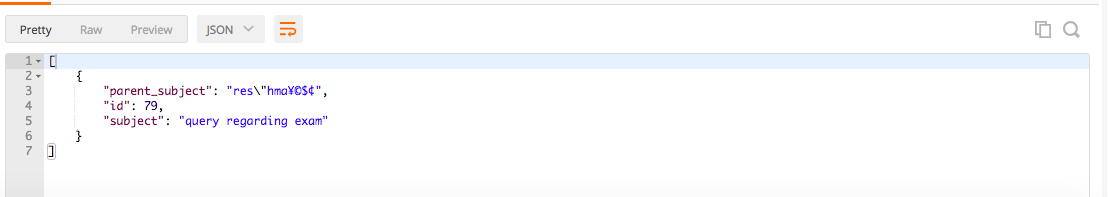
}}

}

RestCall:”POST”: “https://kptickets.appup.cloud/v1/joins”



Returns



## Right Join

The RIGHT JOIN returns all records from the right table (table2), and the matched records from the left table (table1). The result is NULL from the left side, when there is no match.

This is similar to joins, it's simply replace join with right join.

# Custom Fields

Custom Fields are any fields not already present in tables, businesses use custom fields to track extra details and map additional fields.

For implementing this need to have Custom\_Field table and <<Entity>>\_Custom\_Field table. In Entity\_Custom\_Field table actual data will be saved.

Custom\_Field has below fields

Name : Name of Custom Field

Type: Field Type (eg: Number, Date, String)

Description: Describe the Field

Default Value: Default Value

Is Required: Mandatory or not.

Entity\_Custom\_Field has below fields

Entity\_ID

Custom\_Field\_ID

Value

**CRUD Operations**

First all the custom fields needed for the application has to be managed. We can use the general CRUD operations explained in previous sections to add/update/delete/retrieve the custom fields.

**Insert new custom field**

This is used for inserting new custom field in the Custom\_Field table.

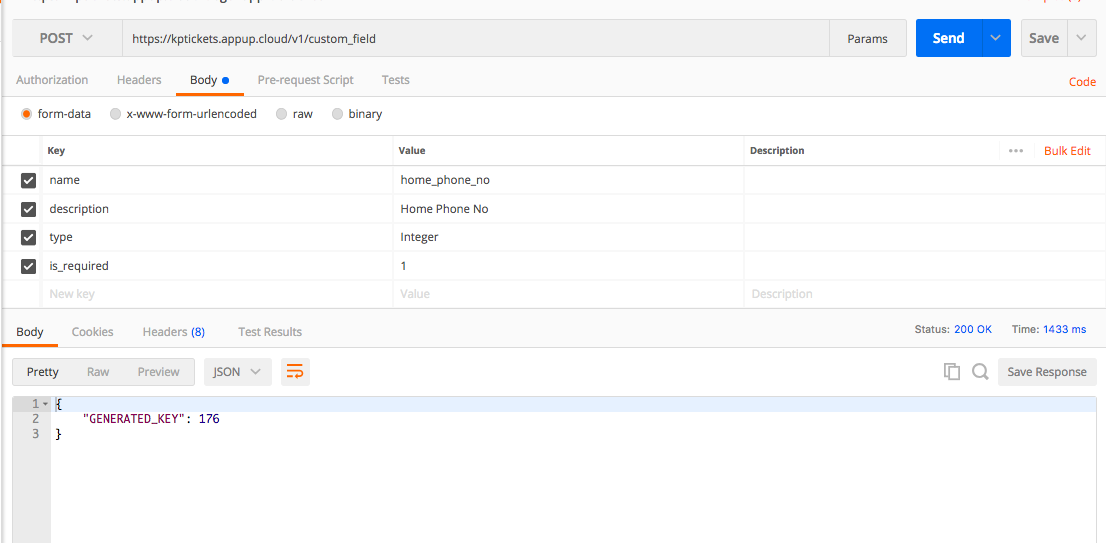
Let us take example for contact entity home phone no is required. We need to create a new custom field home\_phone\_no with integer data type. The values has to be passed as part of request body

Trigger Expression:/v1/cf/{table\_name}

Method: POST

Content-Type: Body

Rest Call:”POST”:”https://kptickets.appup.cloud/v1/cf”



Returns

A 200 response code would be given back with the generated auto incremented sequence id which would serve as primary key of the table as below

{

"GENERATED\_KEY": 176

}

**Deactivate a custom field**

This is to soft delete a custom field associated with all entities. The custom field would not be visible. A put operation may be used to deactivate the custom field

**Add a custom field value to an entity (POST)**

The custom table are like contact\_custom\_field, ticket\_custom\_field, article\_custom\_field.

Trigger Expression: /v1/cf/{table\_name}/{entity\_id}

Method: POST

Content-Type: application/json

request={

"body":{

"Data":[

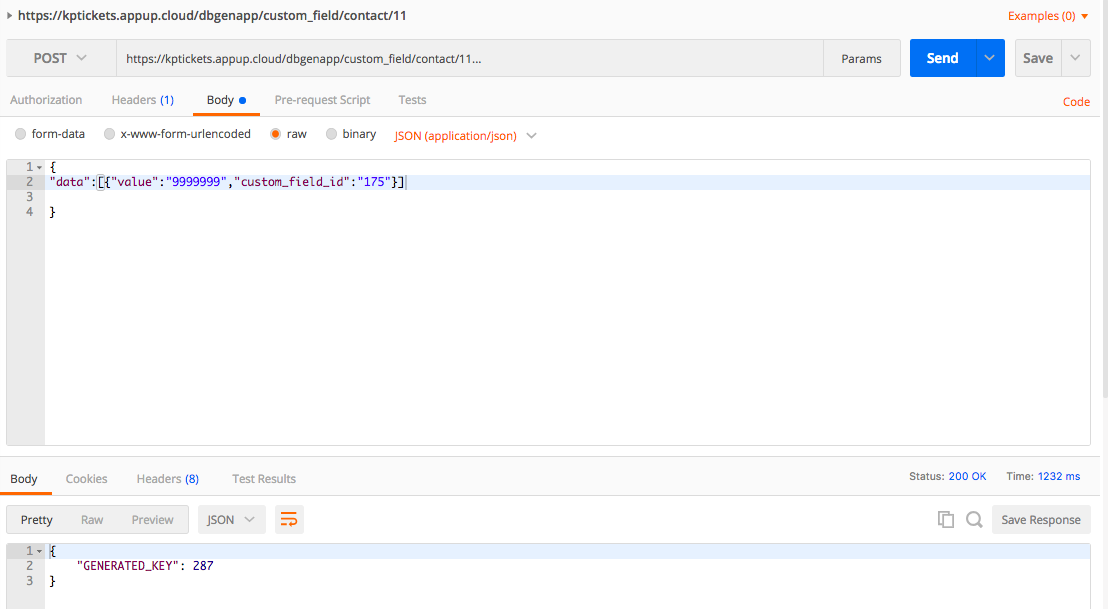
{"value":"9999999","custom\_field\_id":"176"}

]

}

Let us take example where we want to insert a new field alternate phone no for the contact entity for a particular customer that doesn’t come out of the box in the app. First this need to be created as explained in “Insert new custom field section” a record in custom\_field table with name as alternate phone no and type as varchar. As second step user wants to map this field in their contact and specify the alternate phone number

RestCall:”POST”:”https://kptickets.appup.cloud/v1/cf/contact/11”



Returns

A 200 response code would be given back with the generated auto incremented sequence id which would serve as primary key of the table as below

{

"GENERATED\_KEY": 176

}

**Update a custom field value for an entity(PUT)**

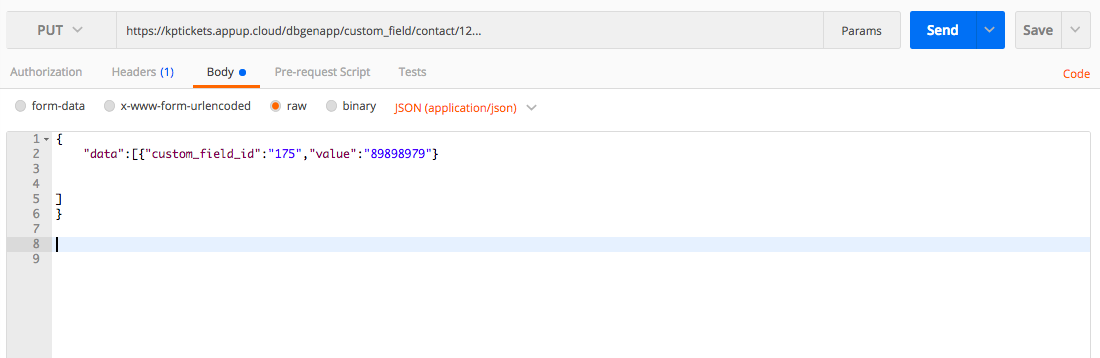
Appup will update the data of the custom field data.

Trigger expression: /v1/{table\_name}/{entity\_id}

Method:PUT

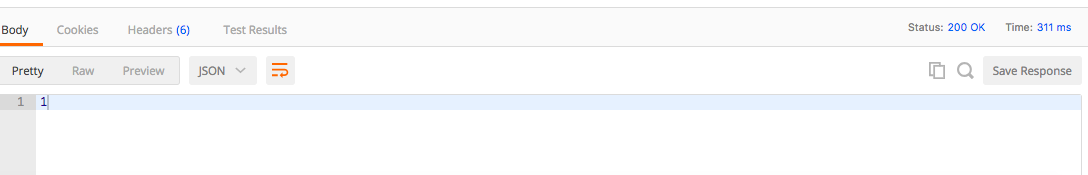
Let’s take an example where required to update the data of home phone no in custom field table. The postman example is given here as reference. The values has to be passed as part of request body.

Rest Call:”PUT”:”https://kptickets.appup.cloud/dbgenapp/cf/contact/11”



Returns

The number of rows updated would be returned



**Remove a custom field (DELETE)**

This api will be used to delete a custom field associated with all the entity.

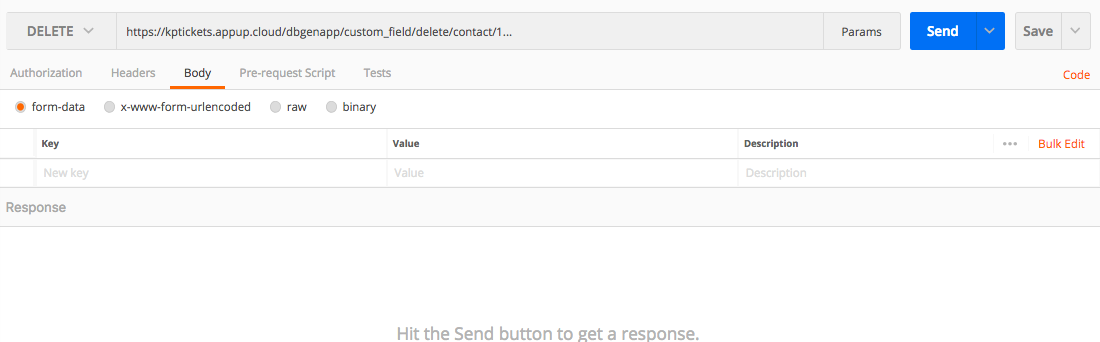
Let us take example where we want remove alternate phone no and work phone no, we will pass the corresponding custom field id values in params.

Trigger Expression: /v1/custom-field/{table-name}/{custom-field-id}

Method: DELETE

Params: Entity Custom Field ID’s

Rest Call:”DELETE”:”https://kptickets.appup.cloud/dbgenapp/custom\_field/contact/1”



Returns

The number of rows deleted would be returned.

**Retrieve all the data with a given entity id(GET)**

This api shall be used to retrieve all the custom fields data associated with the given entity id.

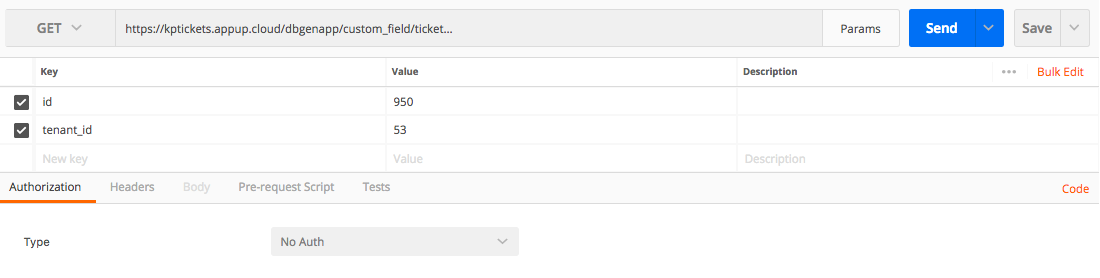
Trigger Expression:/v1/custom\_field/{table-name}/?custom\_field/{table-name}

?id=&tenant\_id=

Method: GET

Rest Call:”GET”:”https://kptickets.appup.cloud/dbgenapp/custom\_field/ticket

?id=950&tenant\_id=53”



Returns:



**Retrieve all the data with a any entity(GET)**

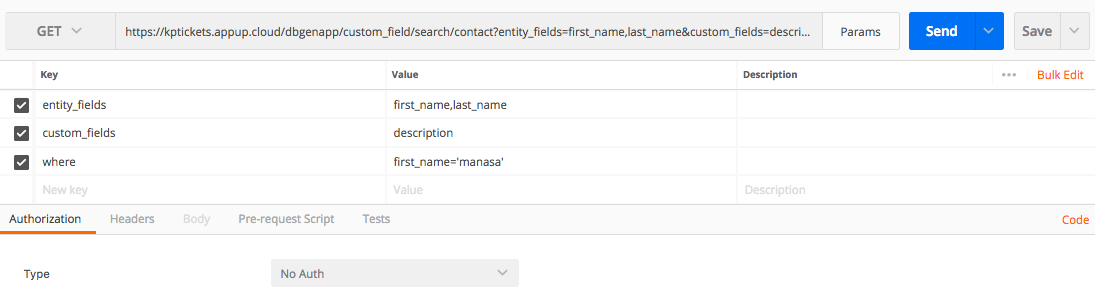
This api shall be used to retrieve all the custom fields data associated with the any entity.

Trigger Expression:/v1/retrieve/{table-name}/?custom\_field/search/{table-name}

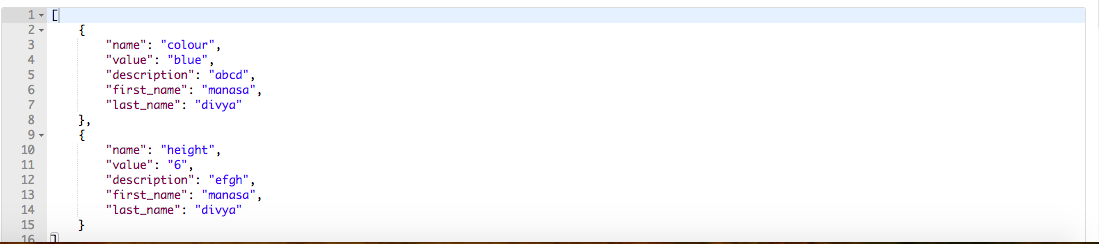
?id=&tenant\_id=

Method: “GET”

Rest Call:”https://kptickets.appup.cloud/dbgenapp/custom\_field/search/contact?entity\_fields=first\_name,last\_name&custom\_fields=description&where=first\_name='manasa'”



Returns:



# Reports - Not yet started

A report are the formatted result of database queries and contains useful data for decision-making and analysis.

**Bar Chart:**  Using this workflow, developers can generate bar charts query, for this need to provide x-axis column and y-axis column, functions(avg, sum,count etc) and joining conditions.

Let us take example of where required monthly wise tickets count. Months in x-axis and ticket counts in y axis to be displayed in graph. So we need to generate query like that.

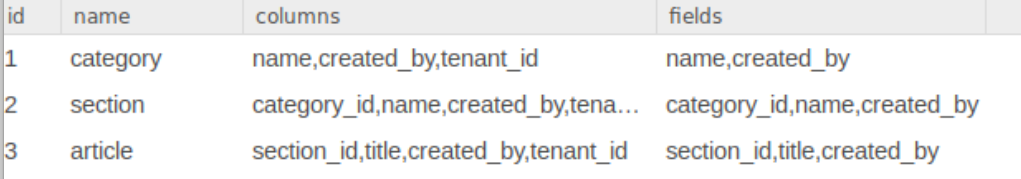
The input data has to be sent as part of the request body as JSON format. Any attribute that is not needed for a particular operation should not be included.

# Default Values

This api may be used when a new customer signs up or a new user is invited.

Pre-processing Steps:

Step1: Need to create table “**domain\_default\_data**” with columns name,columns and fields.



Names → Application related default tables

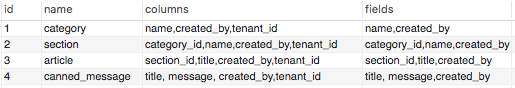
Columns → Default tables related columns names to be inserted.

Fields → Get the fields form default table.

Let us take example of Support.ly application,in that canned\_message table has required default data, So that we will insert new record as below

|  |  |  |
| --- | --- | --- |
| Name | Columns | Fields |
| canned\_message | title, message, created\_by,tenant\_id | Title, created\_by, message |

Domain\_default\_data table:

****

Step 2: Need to insert required rows in above tables with tenant\_id/domain\_id is ‘0’.

As below mentioned format required data should be updated with tenant\_id is 0 in canned\_message table.

|  |  |  |
| --- | --- | --- |
| title | message | tenant\_id |
| footer greeting | Im hoping that I was able to answer your queries, <requester\_name>. Please feel free to get back to us if you still have any questions | 0 |

This below appup will be used add default settings.

## New account sign up

When new customer signs up for an app. There would be some default data to be setup for the new account so when he logs into the application, he sees them.

For example when a user signs up for Supportly, default group has to be created. Default tags has to be created.

* List of default-table - table-names to copy for account level and user level
* For each table, copy all records for that table (user-id=0) to the new account/user

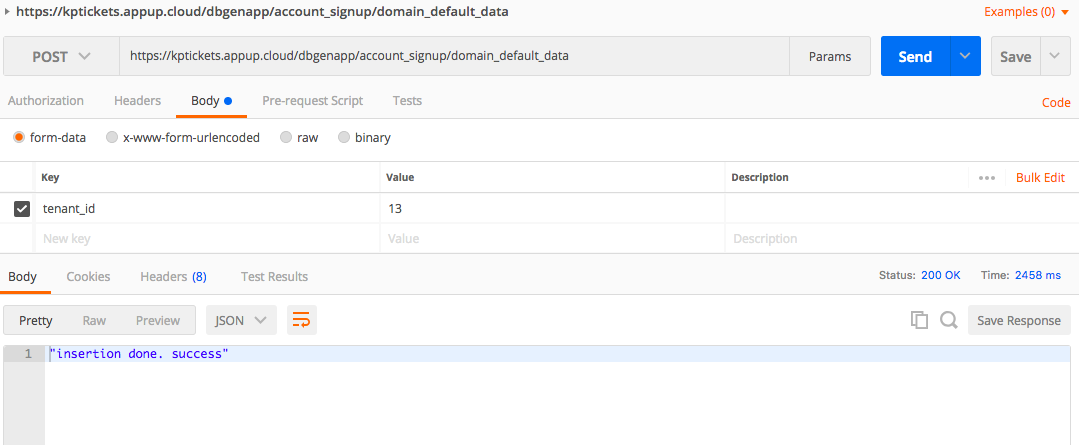
On user creation, there are certain tables and rows, we need to create and replicate.

* Handlebars (Should support handle-bars)

Trigger Expression: /v1/account\_signup/{default\_tablename}

Method: POST

Rest Call:”POST”:”https://kptickets.appup.cloud/dbgenapp/account\_signup/domain\_default\_data”

Returns

A 200 OK response code would be returned along with “insertion done success” with this message.

## Invite new user

Need to understand the scope and example. Coming soon

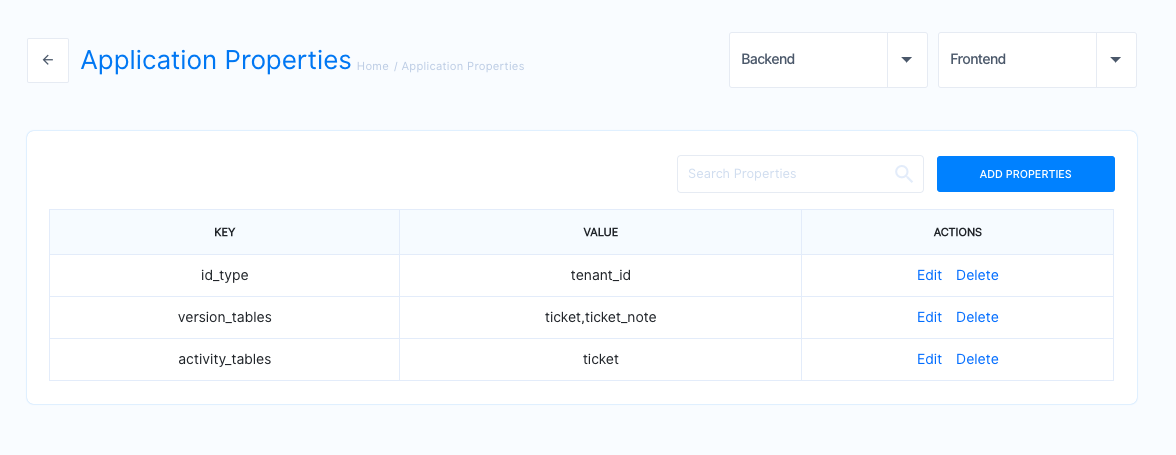
# Activity

Stores the activities for a certain entity automatically. Audit logs.

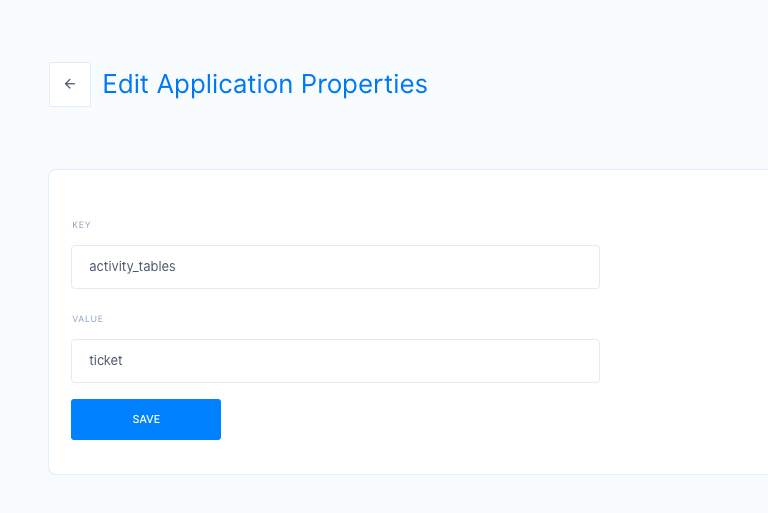
This appup contains audit messages that indicate data activity relating to specific data in the application. The Developer/user needs to identify which tables are required to capture the audit logs.

Setup for Activity

Need to add new key “activity\_tables” in application properties settings like below screenshot.



Activity required tables to be added in value like below screenshot.



Next steps will be regular crud operations.

# 

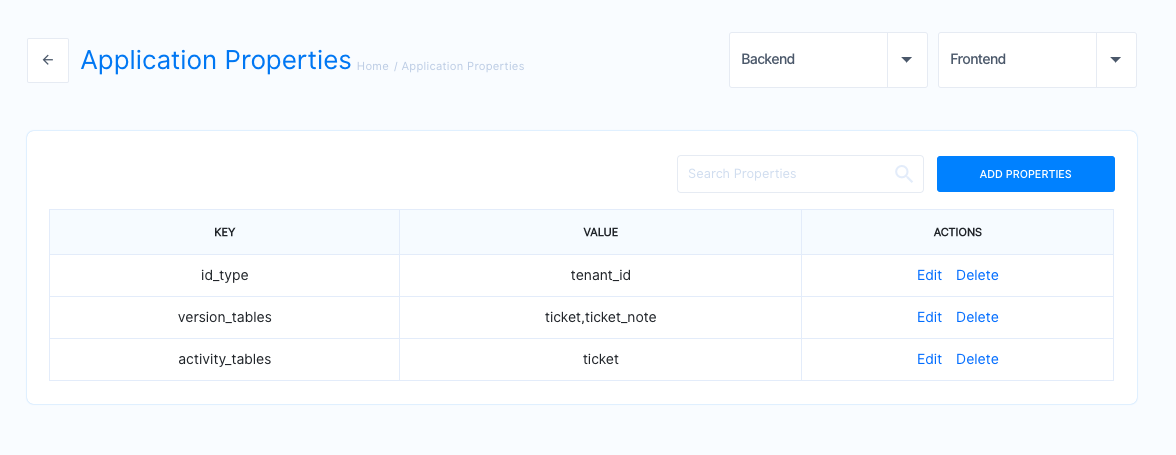
# Versioning

Typically, for versioning or storing historical data you do one of two (or both) things. You have a separate table that mimics the original table + a date/time column for the date it was changed.

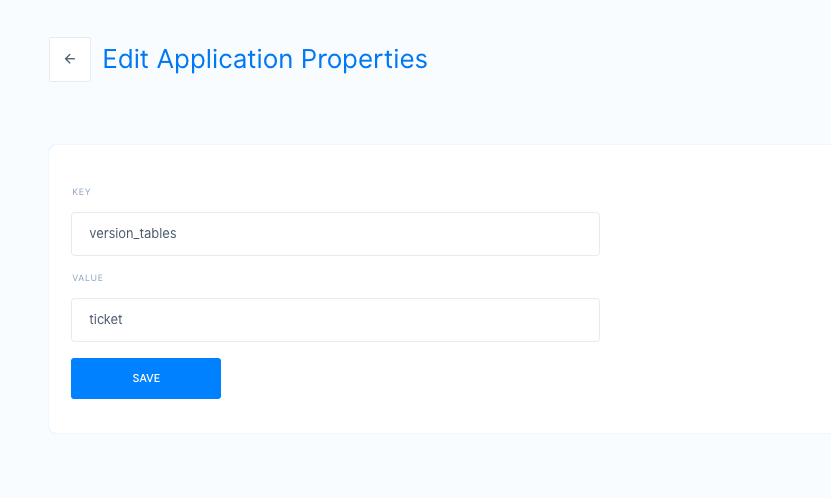
Stores the version for a certain entity automatically.

Setup for Version

Need to add new key “version\_tables” in application properties settings like below screenshot.



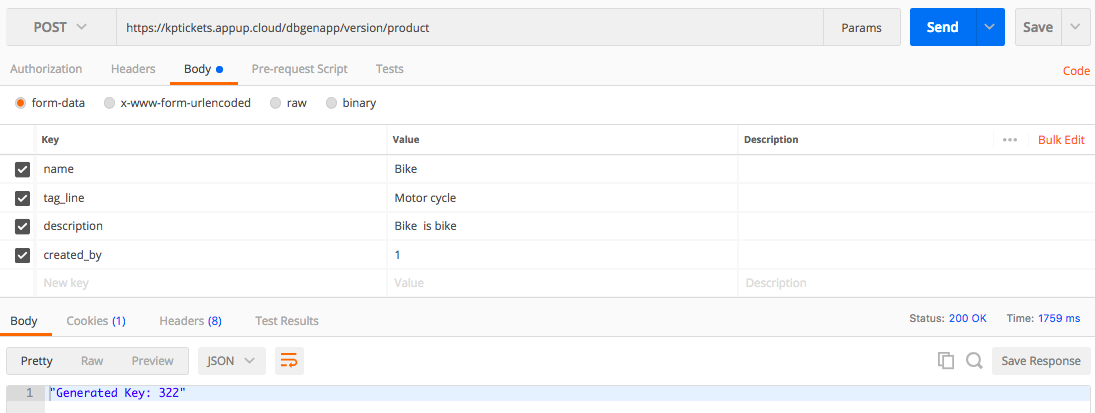
Version is required tables to be added in value like below screenshot.



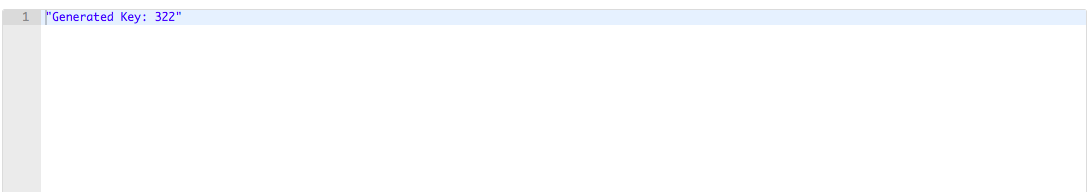
Trigger Expression:/v1/{table-name}

Method: POST

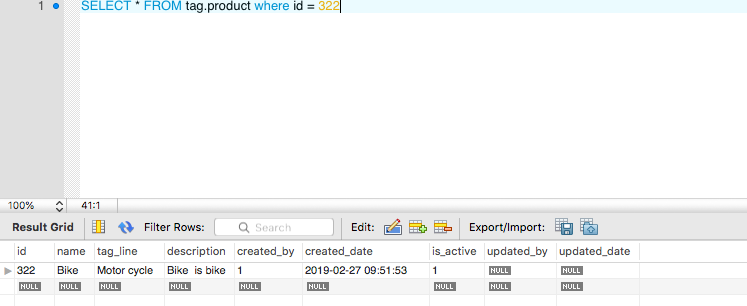
Rest Call:”[https://kptickets.appup.cloud/dbgenapp](https://kptickets.appup.cloud/dbgenapp/)/product”



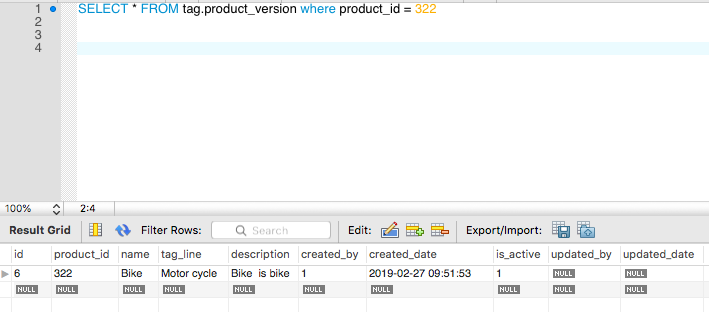
Returns



Product Table data



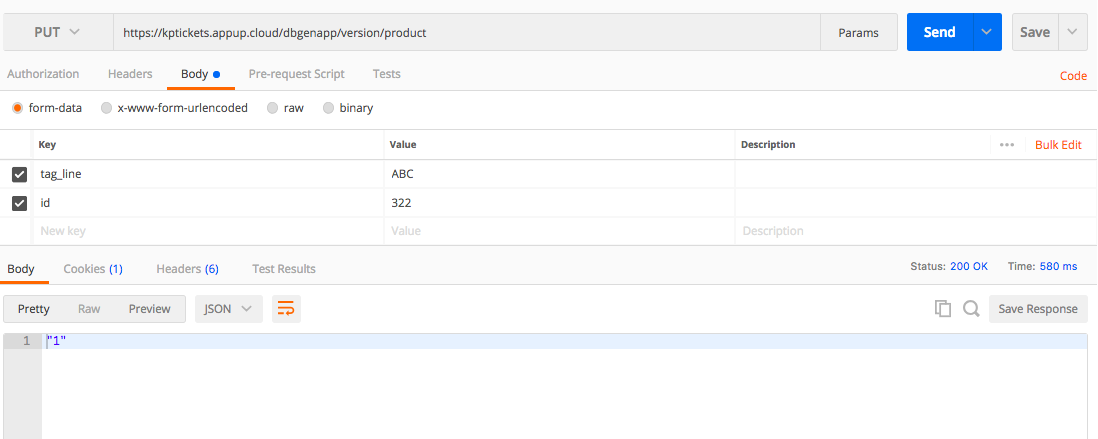
Product\_version Data

Update

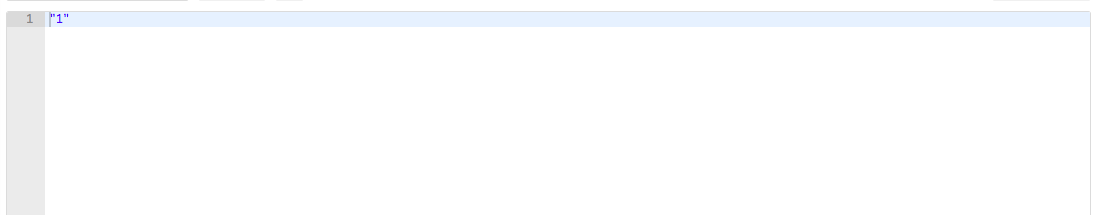
Trigger Expression:/v1/{table-name} (\*\*\*This is temporary tigger in future trigger expression is /v1/{table-name}

Method: PUT

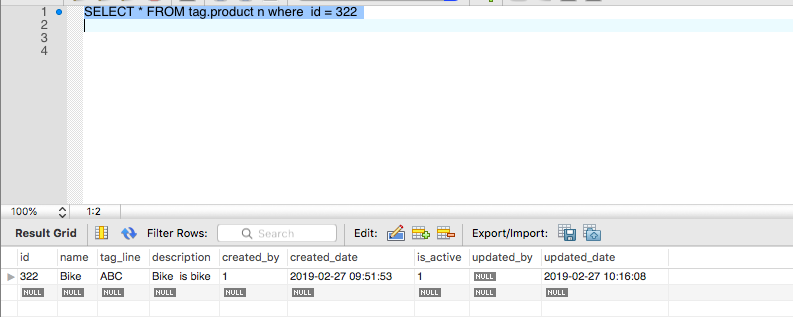
Rest Call:”PUT”:”https://kptickets.appup.cloud/dbgenapp/product”



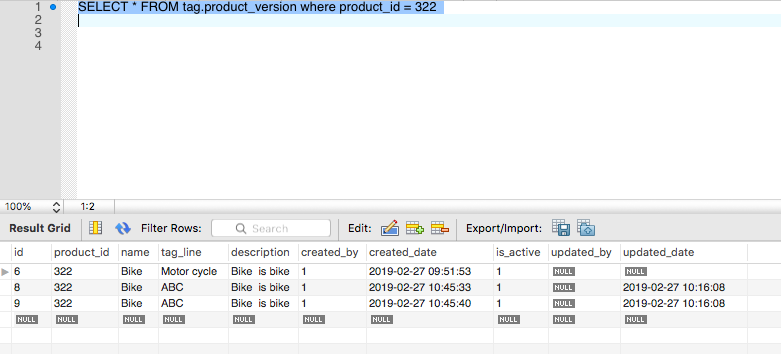
Returns



Product Table



Product\_Version Data:

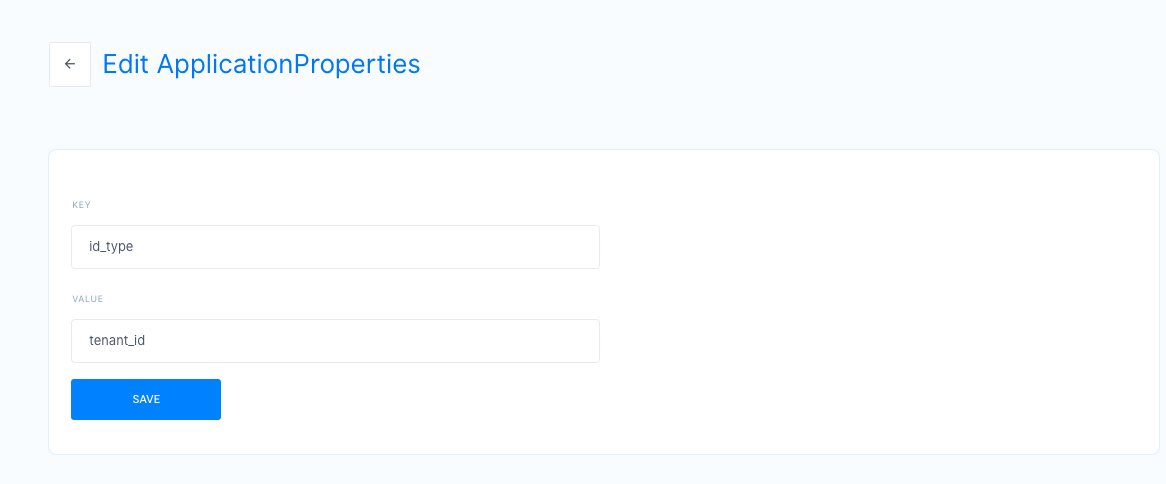


# Security - JWT

Using the JWT, the generics can be used to restrict operations related to only the users who are authenticated in JWT.

**Configuration of Domain ID or tenant id:**

For configuration of domain\_id or tenant\_id, we need to add new variable in app properties, Please find below screenshot for your reference

****

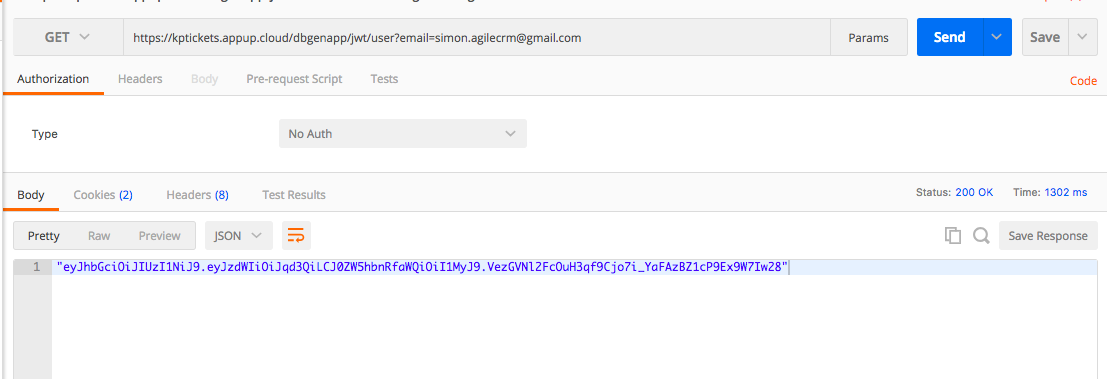
**Token Generation**

Let us take example of generating token for domain id. Below rest call will generate JWT token.

Trigger Expression:/v1/jwt/{table-name}

Method: GET

RestCall:”GET”:”https://kptickets.appup.cloud/dbgenapp/jwt/user?email=simon.agilecrm@gmail.com”



Returns:



## Insert table with JWT

## The browser makes a POST request to the server that contains the domain information. The server responds with a cookie, which is set on the user’s browser, and includes a session ID to identify the domain.

This endpoint creates a row with JWT in the mentioned table name using Appup

Trigger Expression : /db/{table\_name}

Method: POST

Example:

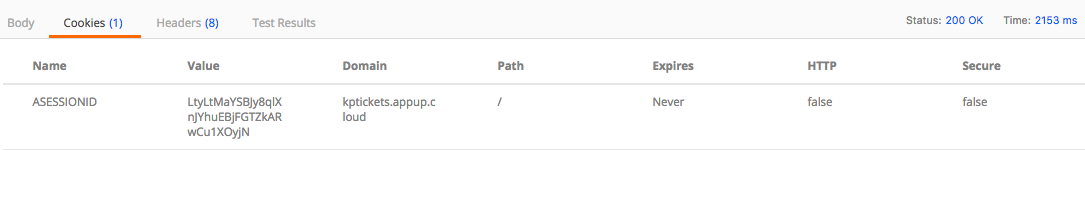
Let’s take an example where we want to insert into **ticket** table for **fields** subject,category etc

The values has to be passed as part of request body.

The postman example is given here as reference.

RestCall:”POST”:”https://kptickets.appup.cloud/dbgenapp/db/ticket”

## JWT Filter



# DESCRIBE

Describe is used to describe the metadata of table. This appup is describe the structure of a table.

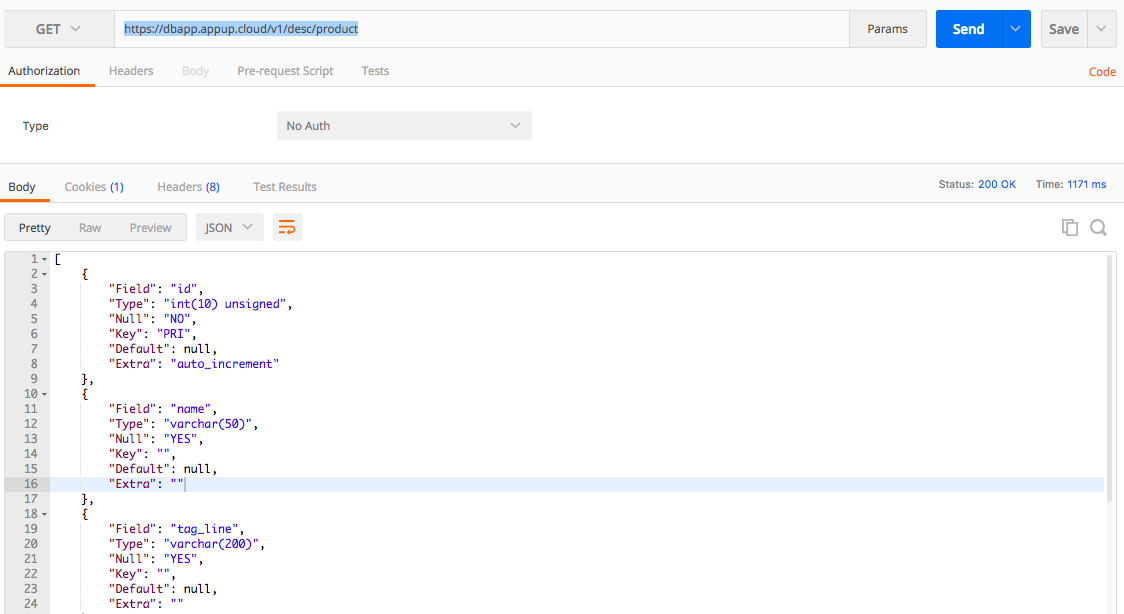
Trigger expression /v1/desc/{table\_name}

Method: GET

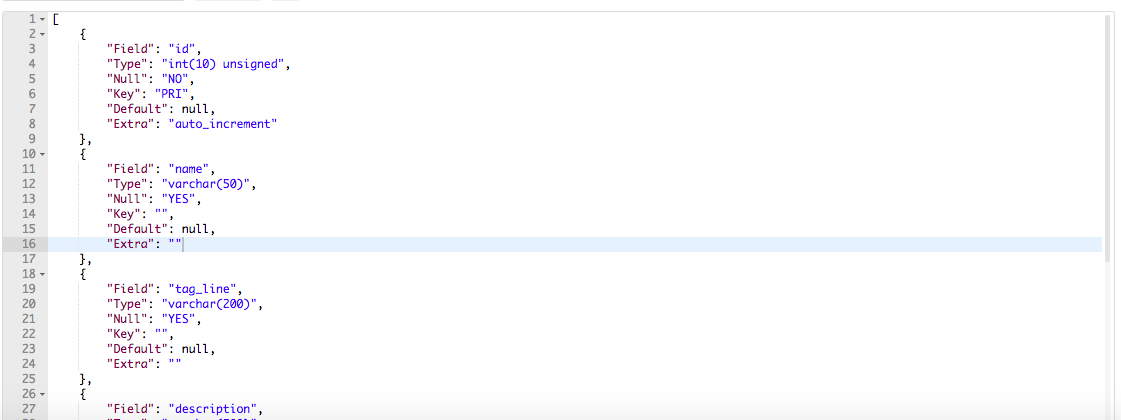
Example:

Suppose product table has some columns named id, name,tag\_line, description and some are of can contain null values. So desc appup shows the structure of table which include name of the column, data-type of column and the nullability which means, that column can contain null values or not. Find below screenshot for your reference

Rest Call:”GET”:”<https://dbapp.appup.cloud/v1/desc/product>”



Result:



# Search in multiple entities

Let you do the search for multiple entities once at time, this appup will useful.

Trigger expression: v1/search

Method: POST

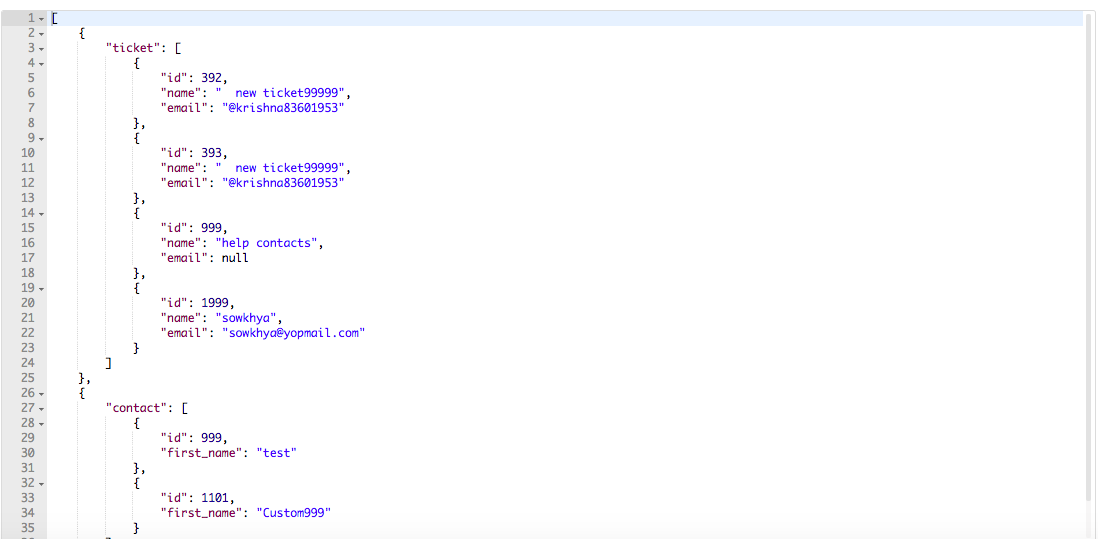
Example:

For example, if you want to search ticket table and contact tables once, let you do this quickly and easily by using this appup. The input data has to be sent as part of the request body as JSON format. Find below screenshot for your reference

Rest call: “POST:https://dbapp.appup.cloud/v1/search”



Result:



# Conflict Check

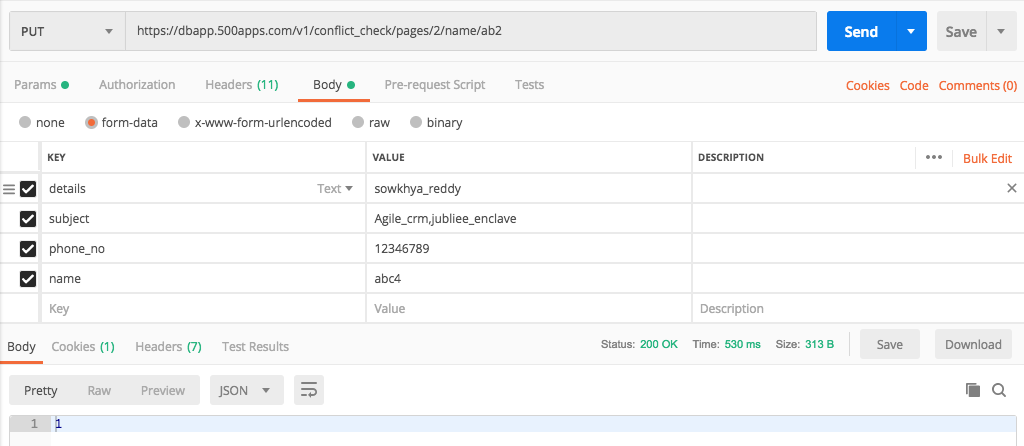
This api does both the conflict check and update operation on the given fields in the body of the request. When conflict based on the field\_type and value, it throws error else it performs update operation on the given table and id

Trigger expression: /cf-cks/{table\_name}/{id}/{field\_type}/{value}

Method: PUT

The required update values have to be passed as part of request body.

Rest Call: “PUT:”https://dbapp.500apps.com/v1/cf-cks/pages/2/name/ab2”



Return: If field value matches “conflict” will come otherwise no. of records updated result will come.

# Checks

This api will check at both tenant and user level. If the count is non-zero, it will return true else return false.

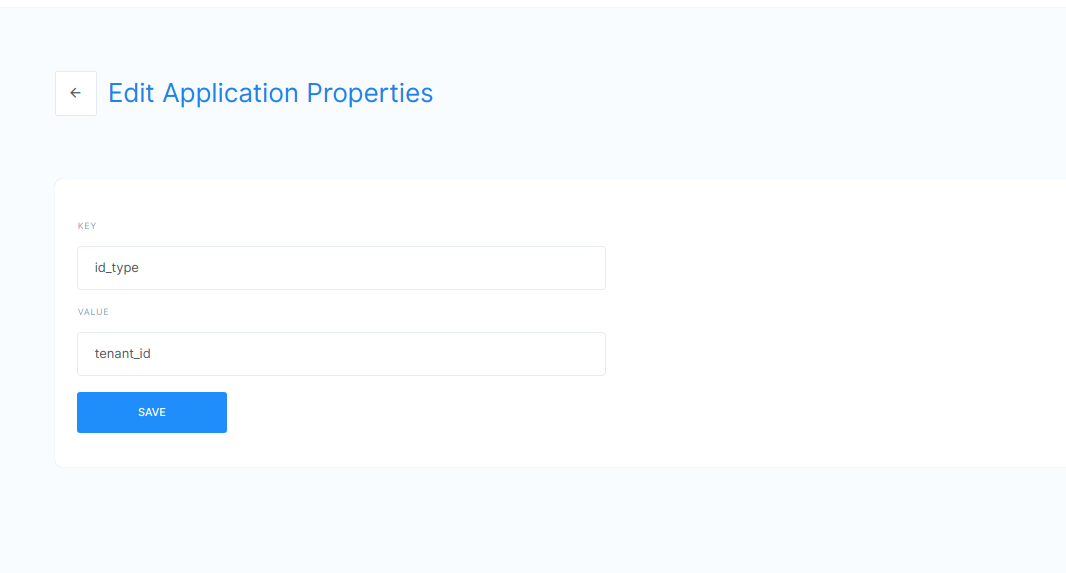
Trigger Expression: /cks/count/{check\_type}/{entity}

Method: GET

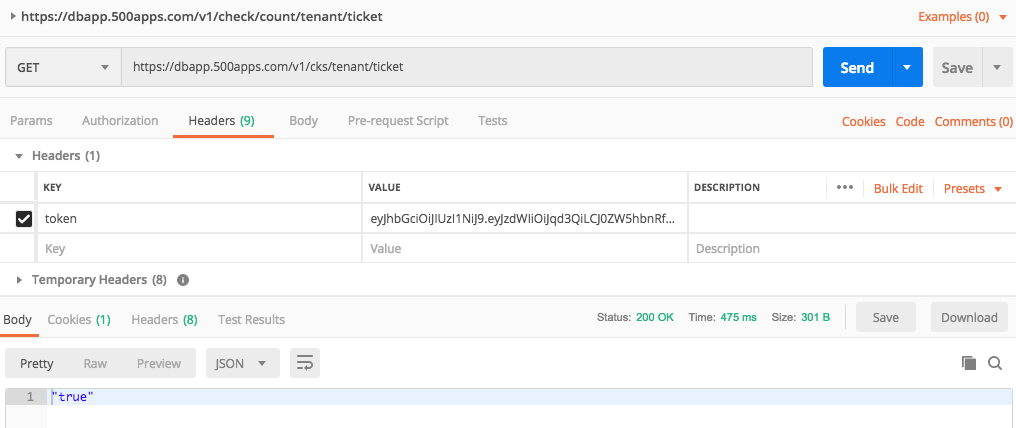
Tenant Level

Need to define **id\_type** in application properties and set the column name based on which you want to check at tenant level in value.

a



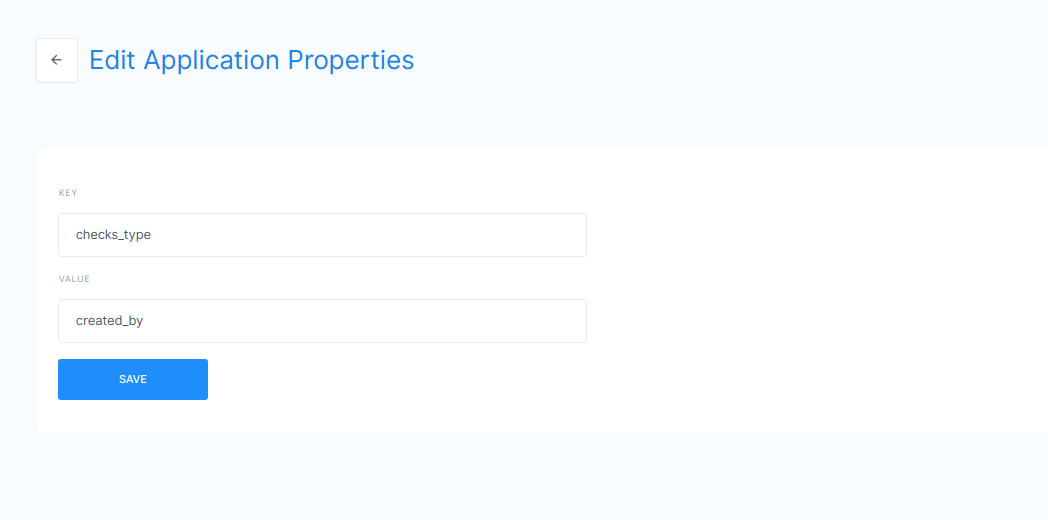
Rest Call:”GET”:” [https://dbapp.500apps.com/v1/cks/tenant/ticket](https://dbapp.500apps.com/v1/check/count/tenant/ticket)”



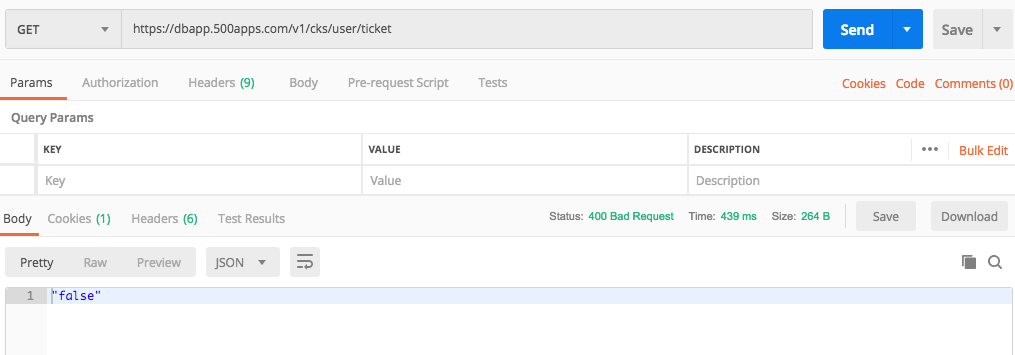
Returns:If the count is non-zero, it will return true else return false.

User Level

Need to define **checks\_type** in application properties and set the column name based on which you want to check at user level in value.



Rest Call:”GET”:”[https://dbapp.500apps.com/v1/cks/user/ticket](https://dbapp.500apps.com/v1/check/count/user/ticket)”



Returns:If the count is non-zero, it will return true else return false.

# JSON Search

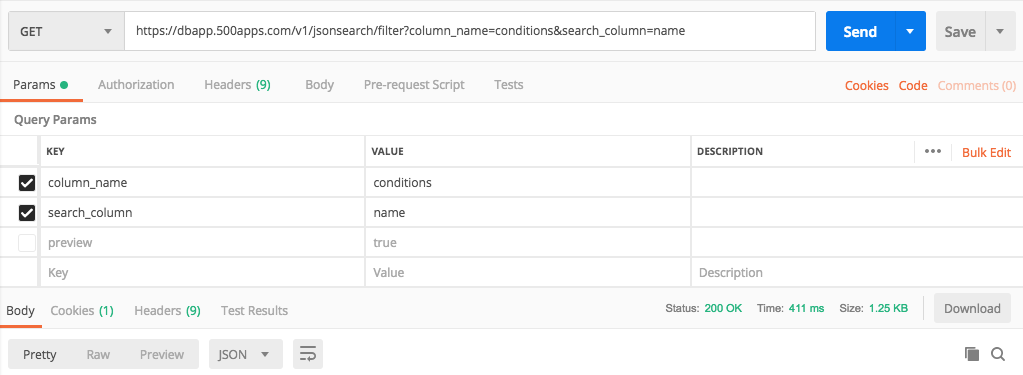
This api returns the data to a given string in a JSON column.

Trigger Expression: /json/search/{table\_name}

Method: GET

If the JSON column contains multiple occurrences of the same string, we will use below rest call. Find below screen for your reference.

Rest Call:GET:”[https://dbapp.500apps.com/v1/json/search/filter?column\_name=conditions&search\_column=name](https://dbapp.500apps.com/v1/jsonsearch/filter?column_name=conditions&search_column=name)”

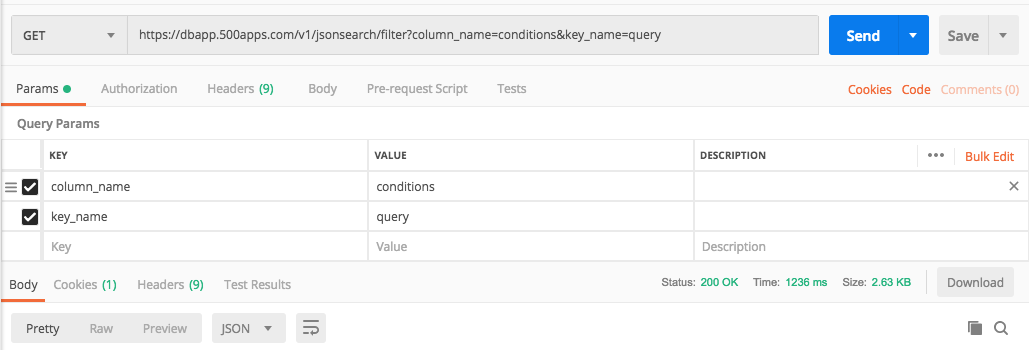


Return:

The paths of all occurrences are returned. If there’s more than one path, they’re auto-wrapped as an array.

If you require specified key in JSON document, we will use below rest call. Find below screen for your reference.

Rest Call:GET: “<https://dbapp.500apps.com/v1/jsonsearch/filter?column_name=conditions&key_name=query>”

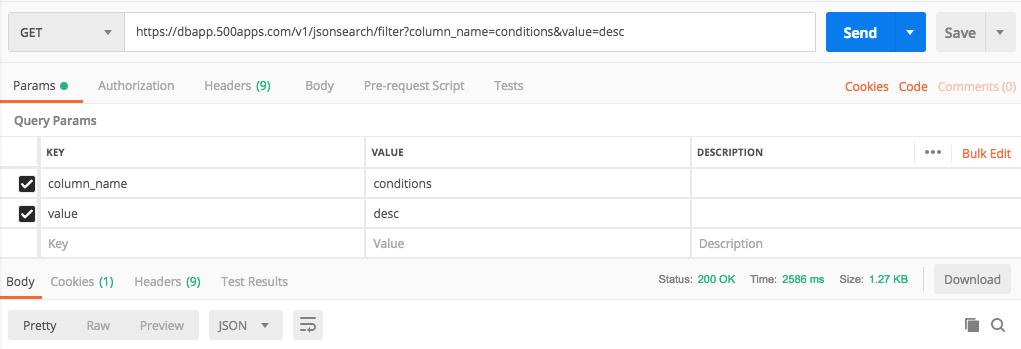


Return:

The paths of all occurrences in key names are returned.

If you require specific value in JSON document, we will use below rest call. Find below screen for your reference.

Rest Call:GET: “[https://dbapp.500apps.com/v1/jsonsearch/filter?column\_name=conditions&value=](https://dbapp.500apps.com/v1/jsonsearch/filter?column_name=conditions&key_name=query)desc”



Return:

The paths of all occurrences in values are returned.

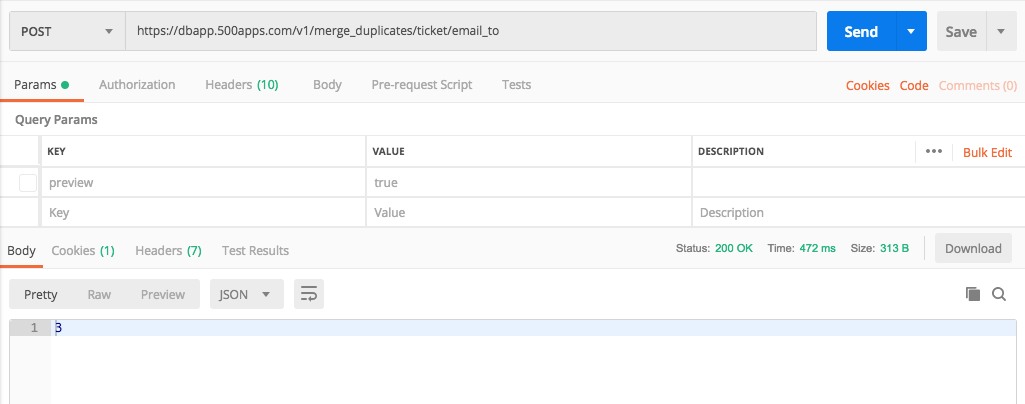
# Merge Duplicates

When you have multiple names of a same person and each name of that person has different contact number saved in your data, you may want to remove the duplicate names from the table this api will use.

Trigger Expression:merge\_duplicates/{table\_name}/{column}

Method: POST

Rest Call:”POST”: “https://dbapp.500apps.com/v1/merge\_duplicates/ticket/email\_to”



Returns:

No. of records merged will be displayed.

# IP LOCATIONS

This api will fetch country\_code, Country\_name, region\_name, city\_name, latitude, longitude, zip\_code, time\_zone column data from the ip\_location table.

If the provided ip address data is not found in the table , the separate rest call will be used to fetch the related data which is also fixed in the workflow itself.

Trigger Expression : ip\_locations

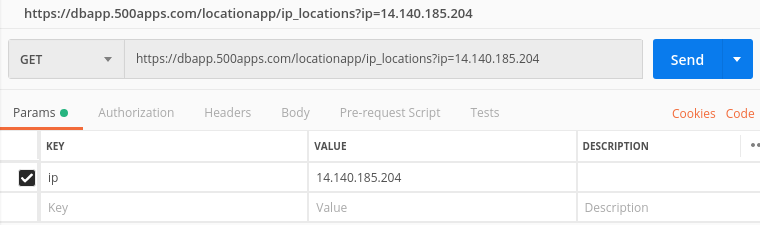
Method : Get

Example : Let’s take an example where ip address **14.140.185.204** is passed through params to get all the required data.

The post\_man example is given below for reference :

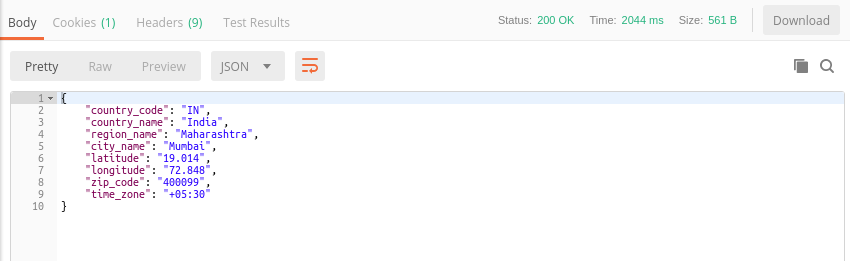
Rest\_call:

[**https://dbapp.500apps.com/locationapp/ip\_locations?ip=14.140.185.204**](https://dbapp.500apps.com/locationapp/ip_locations?ip=14.140.185.204)

****

Result :

A 200 response code would be given back with the required attributes data as shown below :

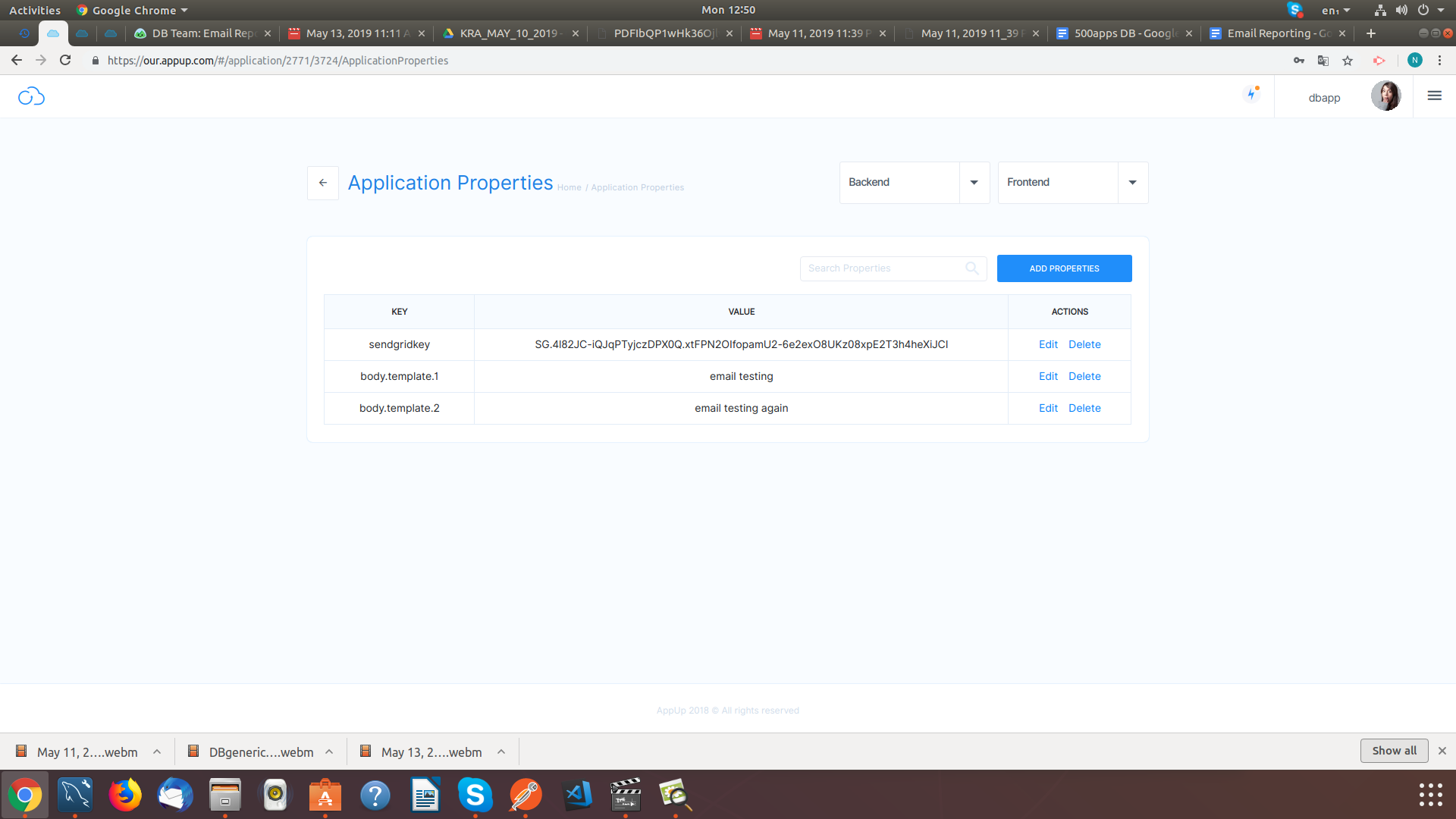


# Email Reporting

Email reporting app is used for send the emails. In this body will append from application properties and the remaining things we can give from frontend.in this application properties we can give the key is like (body.template.1) here body.template is we can give exactly what should be in the application properties.

Trigger Expression:send/{id}

Method:post



Here we can see in the screenshot if we give Trigger Expression be like send /1 the body.template.1 key value (mail testing) will append .if we give send/2 the body.template.2 key value (mail testing again) will append to that corresponding email.

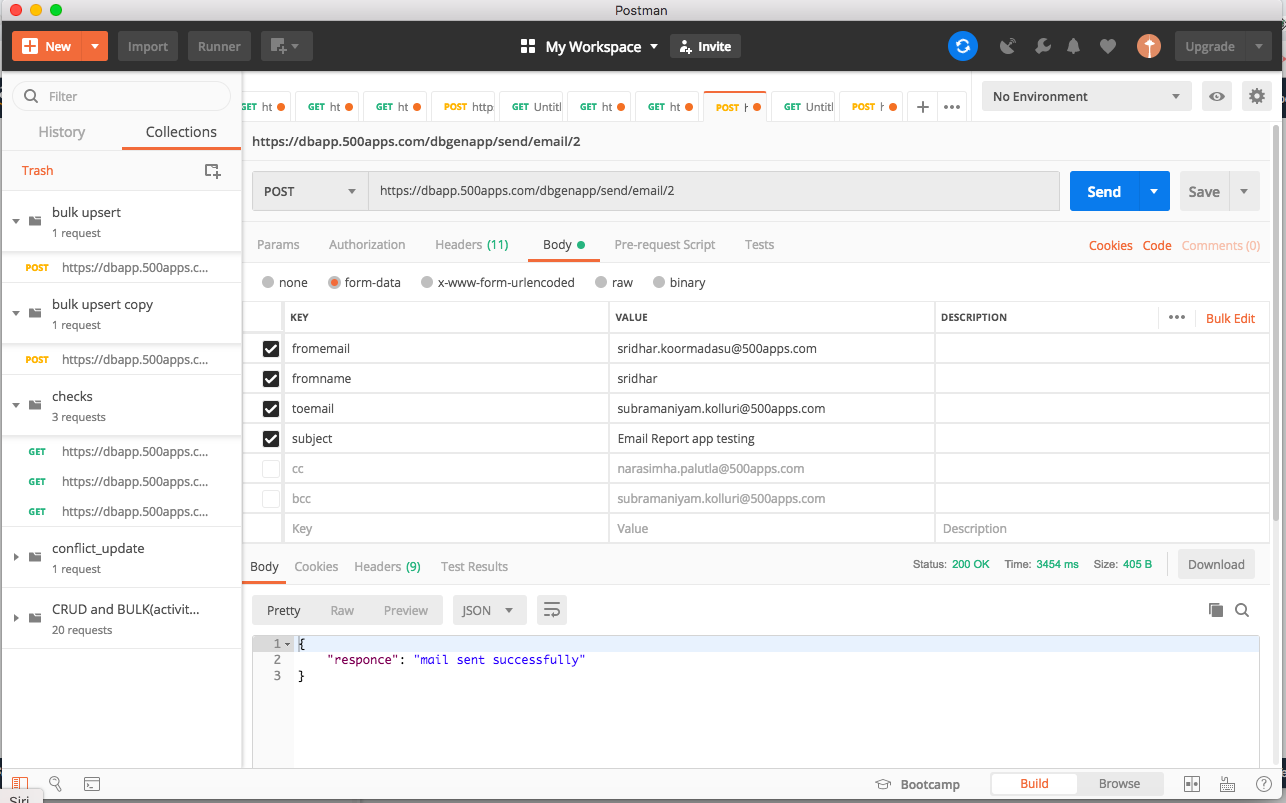
Rest Call:<https://dbapp.500apps.com/emailreporting/send/1>

Example:

Let’s take an example we mentioned from email, to email,subject and from name but we

Sent the data from application properties in appup.The postman example is given here as reference.

￼



Returns: It returns mail sent successfully if we give correct email address else it returns fail to sent.

# Db Form

The main theme of ‘Db Form’ is that we can fetch data of it’s parent(Foreign key table) data. If table have no relation then it fetches only table data what we passed in the postman.In postman we have to specify table name.

TriggerExpression: **/sampleform/user**

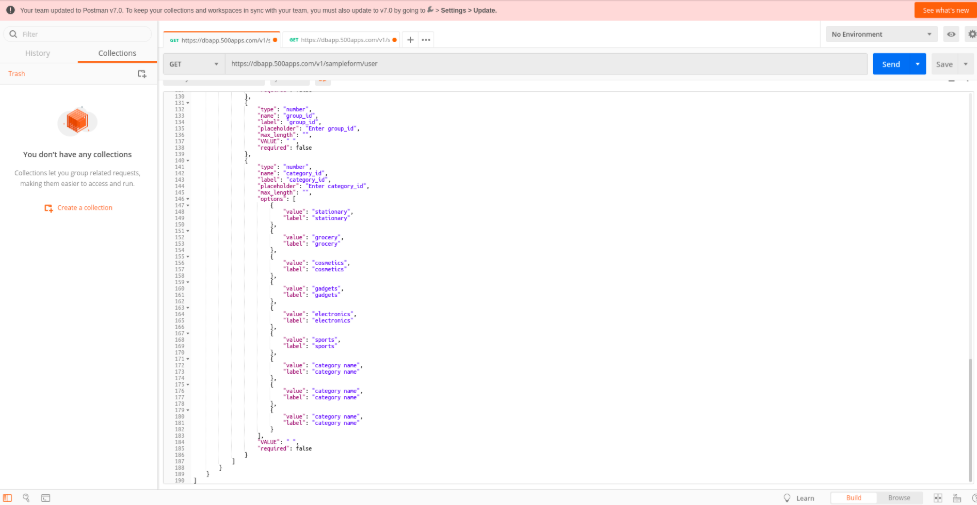
Method: GET

Rest call: https: https://dbapp.500apps.com/v1/sampleform/user

Here is the example how to specify inputs in postman:

Need to pass token in headers.

Returns:



# Query Builder From UI

The main theme of ‘Query Builder From UI’ is that we can filer the data by using where condition and using set conditions like AND and OR. In postman we have to specify input values in json format in the body.

**TriggerExpression:** /querybuilder/{table\_name}

**Rest call:** https://dbapp.500apps.com/dbgenapp/querybuilder/basic

Here is the example how to specify inputs in postman:

{

"fields":"name,price,category",

"data": {

"rules": [

{

"condition": "and",

"id": "name",

"field": "name",

"type": "varchar",

"input": "select",

"operator": "equal",

"value": "gopi"

},

{

"id": "category",

"field": "category",

"type": "varchar",

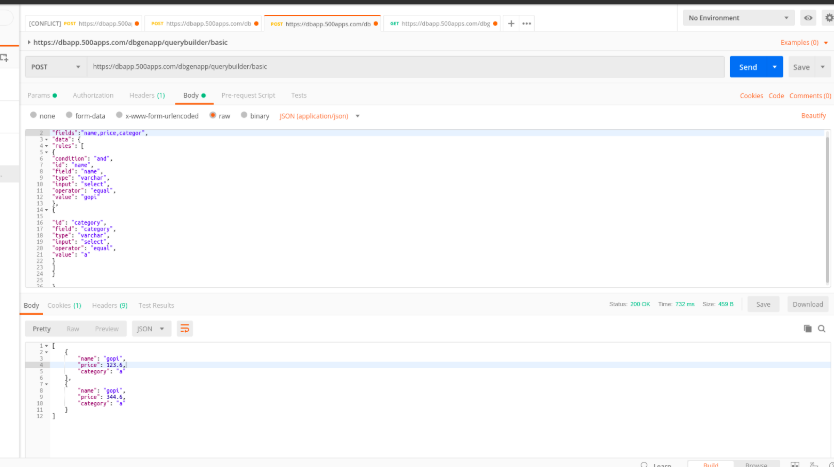
"input": "select",

"operator": "equal",

"value": "a"

}]}

Please find below screenshot



**Returns:**

[

{

"name": "gopi",

"price": 123.6,

"category": "a"

},

{

"name": "gopi",

"price": 344.6,

"category": "a"

}

]

# Access Control Lists (ACL’s)

Access Control Lists in short referred as ACL’s. MySQL uses security based on Access Control Lists (ACLs) for all connections, queries, and other operations that a user may attempt to perform.

In detail ACL’s is the list that controls object permissions, determining which users can executes certain specific tasks.The main theme of ACL’s is to provide security to the data.It’s all the way better to provide only required access rather than giving entire access over the data which may leads to security threats . ACL’s can access with groups , sub groups even for any further extension.

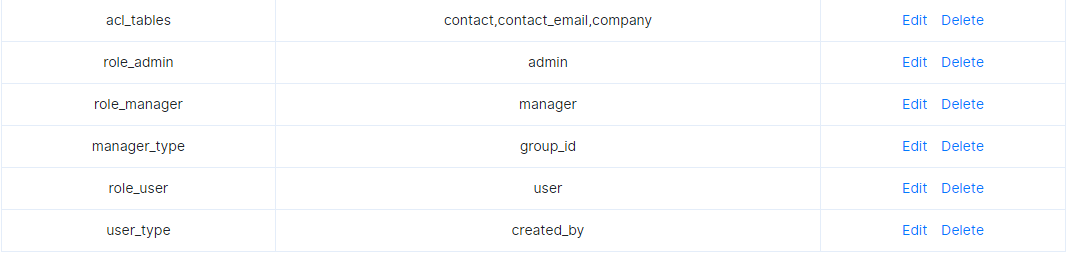
In our scenario we are providing ACL’s on three levels i.e.

* Admin Level
* Manager Level
* User Level

You need to set the following in Application Properties:

1 . The tables on which you need acl’s.

2. Role and the role based dependency column over your schema.



Here the properties

# role\_admin,role\_user,role\_manager can be set with the type of different roles according to the three admin,manager and user levels over your application.

# user\_type,manager\_type can be set by the dependency columns according to the role .

# As the token provided will have the role and id\_type based on the role filter will be appended to fetch the results.

Let us see the postman collection as example :

**API :**  [**https://dbapp.500apps.com/dbgenapp/acls?preview=true**](https://dbapp.500apps.com/dbgenapp/acls?preview=true)

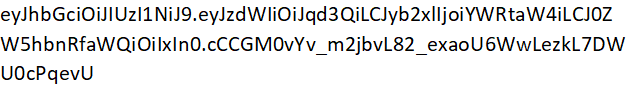
**Trigger Expression :** /acls

**Method** : post

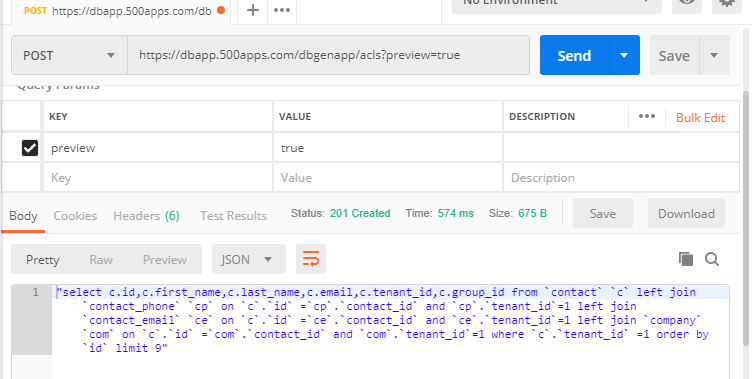
**Admin level** : As admin he can see the entire data over the

application level .

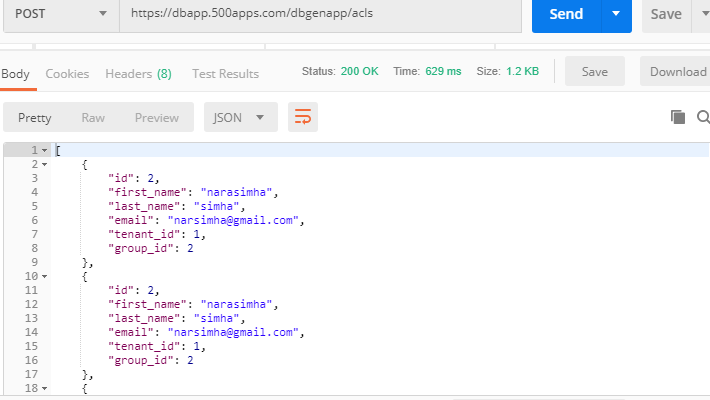
**TOKEN**:



From token role and tenant\_id will be generated:



Data will be fetched based on that filter conditions seen in the above generated query:



**Manager Level**: The manager can access with all the user ‘s data.There may be multiple departments or groups under the manager level ,hence the unique group id/ department id is required

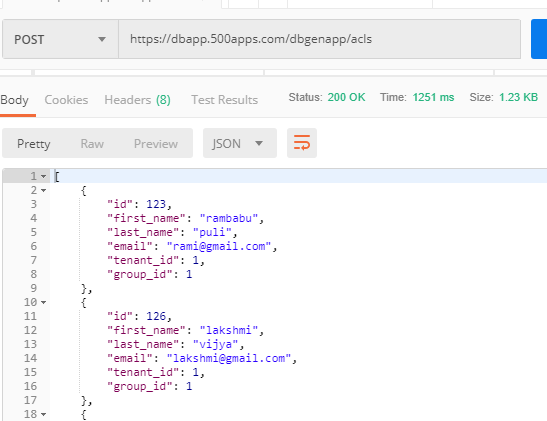
**TOKEN:**

****

From token role,tenant\_id and group\_id will be generated:

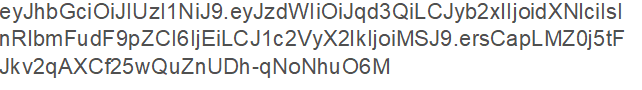


Data will be fetched based on the filter conditions and seen in the above query:

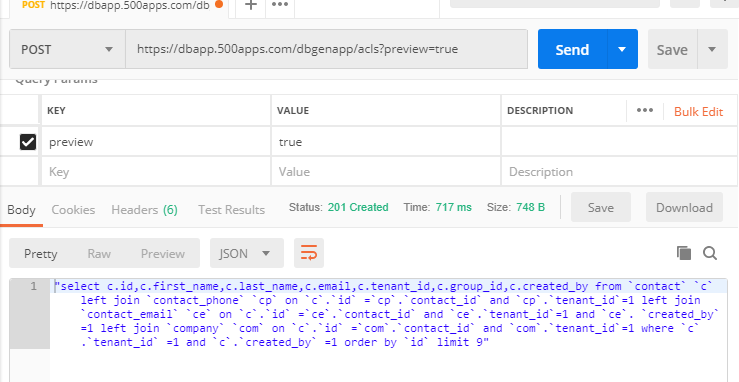


**User Level :** User can have access over his respective data only .

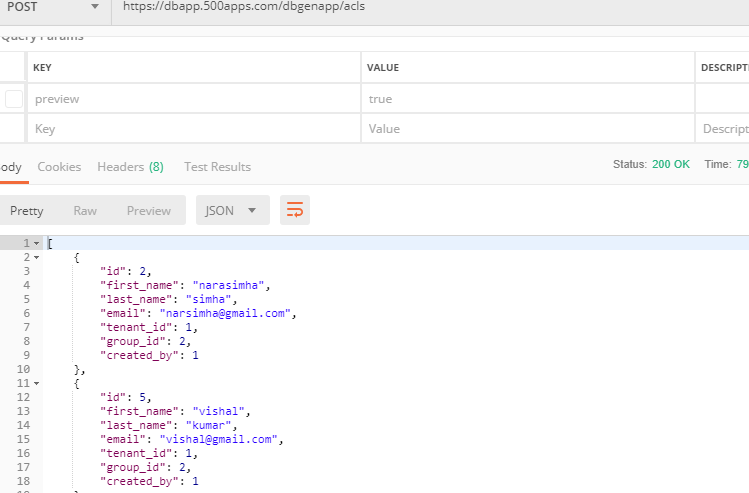
**TOKEN:**

****

From token role,tenant\_id and user\_id will be generated:



Data will be fetched based on the filter conditions appended in the above seen query :



As of now these are the three admin , user , manager levels scenario which can be achieved through appup .

**Advantages :**

**#** ACL’s provides security for the data.

# As discussed initially it is better to provide access only to the required extent rather than giving complete access over the data.

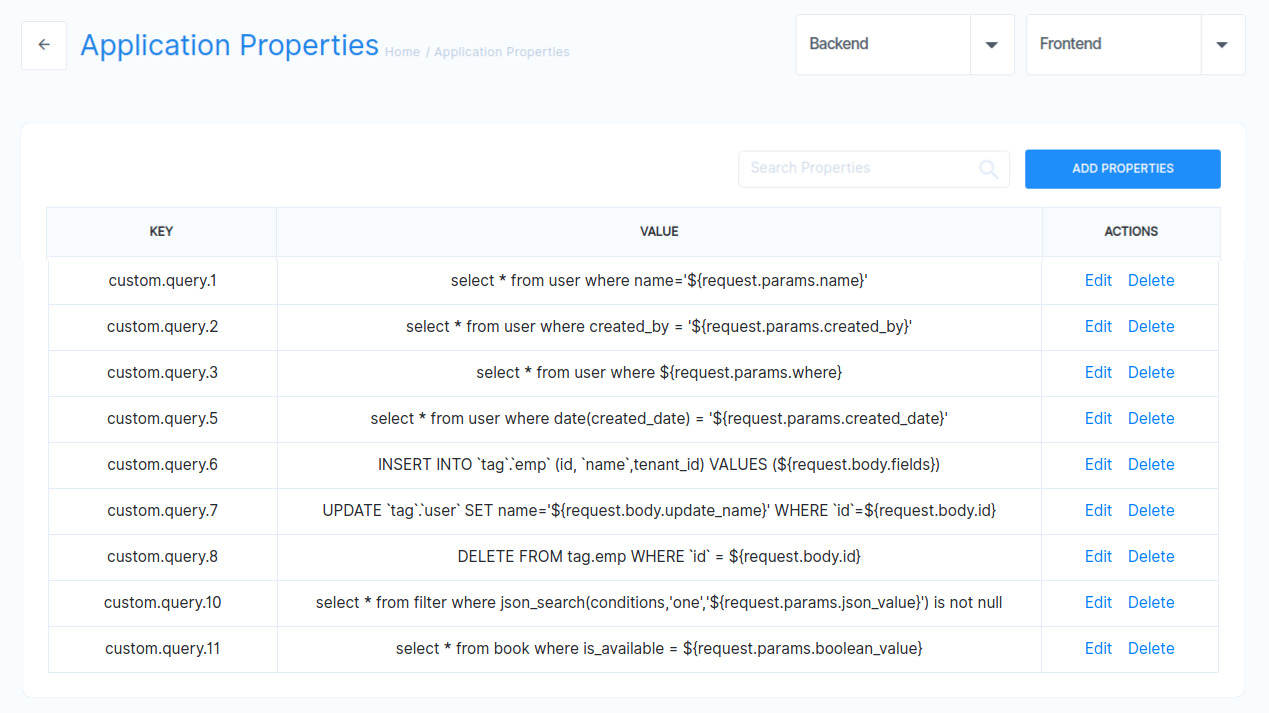
# It is flexible and ease of use for the developers as there is no need to provide the filter conditions through postman rather the suitable ids with respect to the role will be appended directly in the query itself.

# Custom Queries

Main theme :

Users add their custom queries in Application properties.In this scenario we get the results based on customQueryId which we pass dynamically in rest call. In every scenario , id must be unique.

Sum of the example’s (i.e,custom queries) which are added in Application properties.



TriggerExpression: /select/cq/{custom.query.value}

Rest call’s:

Method: **post**

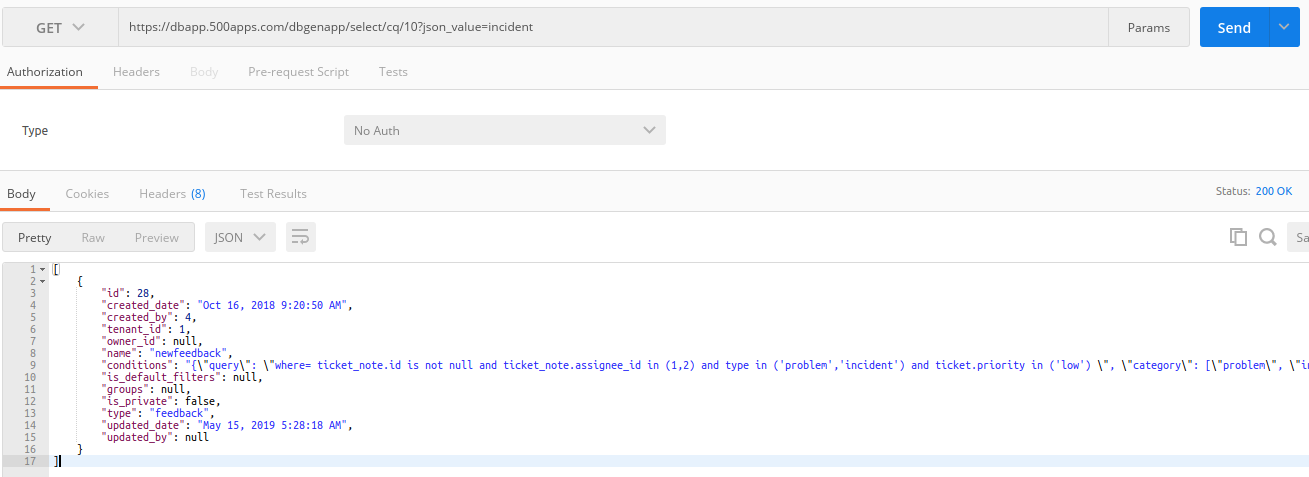
https://dbapp.500apps.com/v1/select/cq/1?name=test

https://dbapp.500apps.com/v1/select/cq/3?where=id=570 and name='Somi'

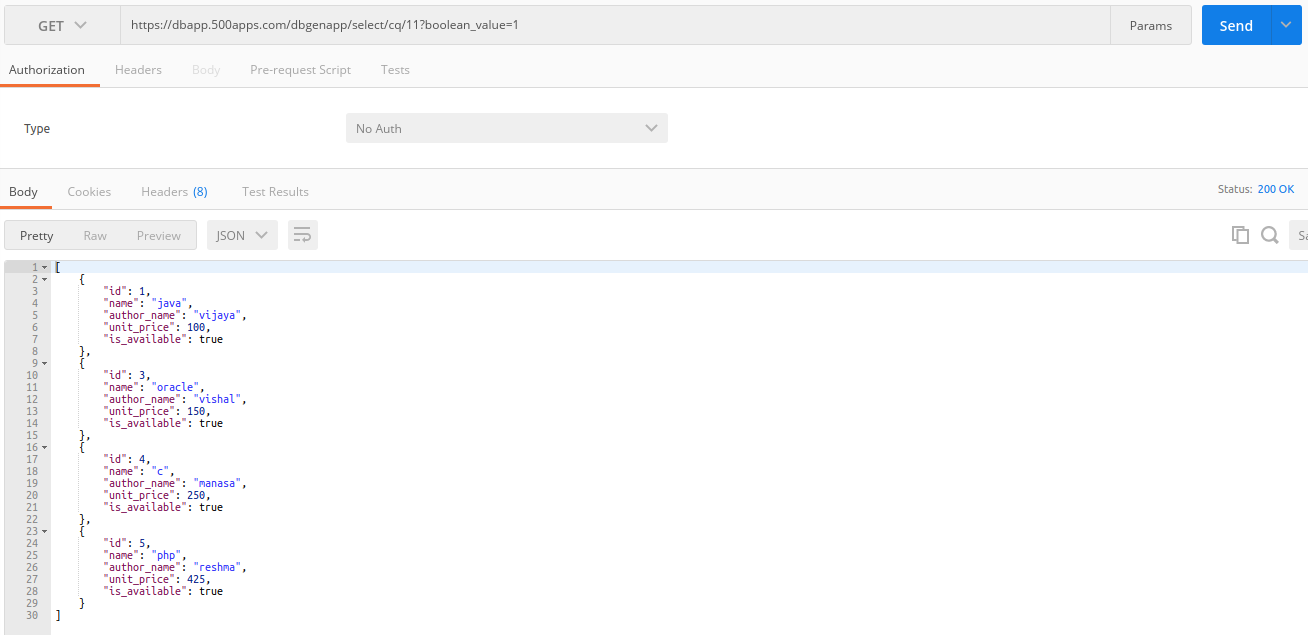
https://dbapp.500apps.com/v1/select/cq/2?created\_by=5

https://dbapp.500apps.com/v1/select/cq/5?created\_date=2018-12-14

https://dbapp.500apps.com/v1/select/cq/10?json\_value=incident



https://dbapp.500apps.com/v1/select/cq/11?boolean\_value=1

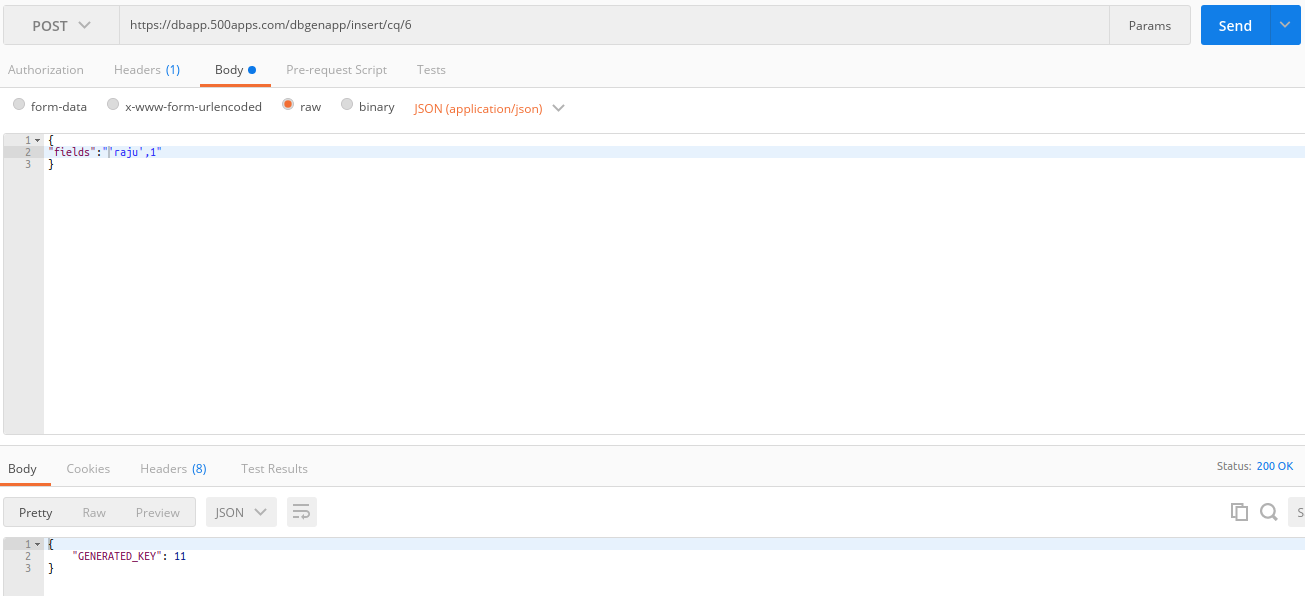


Trigger Expression: /insert/cq/{custom.query.value}

Rest call:

Method: **post**

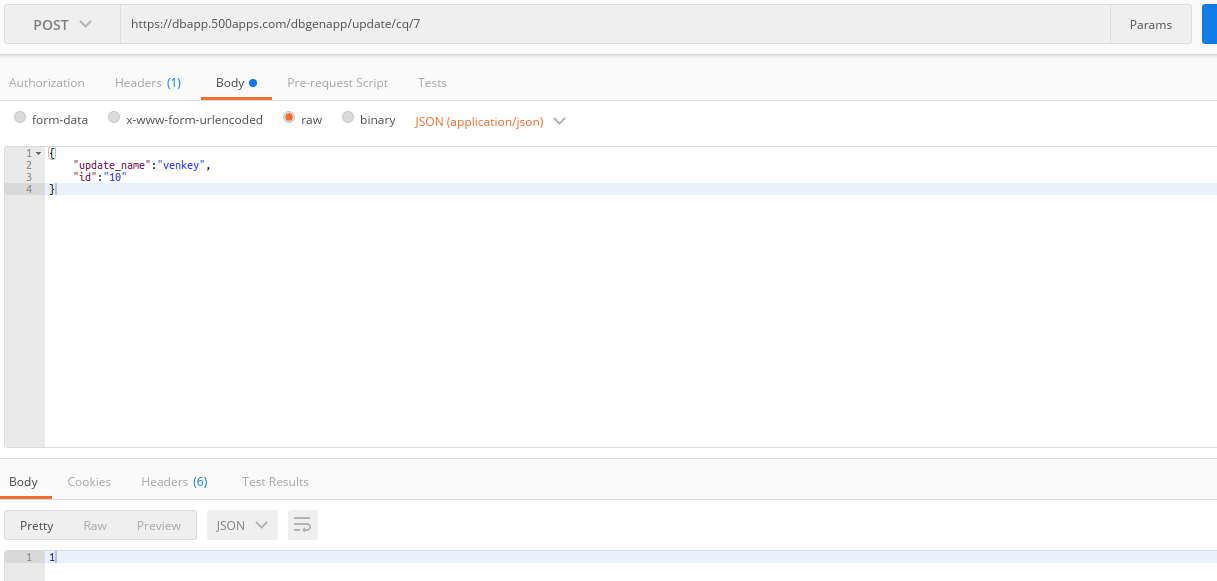
https://dbapp.500apps.com/dbgenapp/insert/cq/6



TriggerExpression: /update/cq/{custom.query.value}

Method: **post**

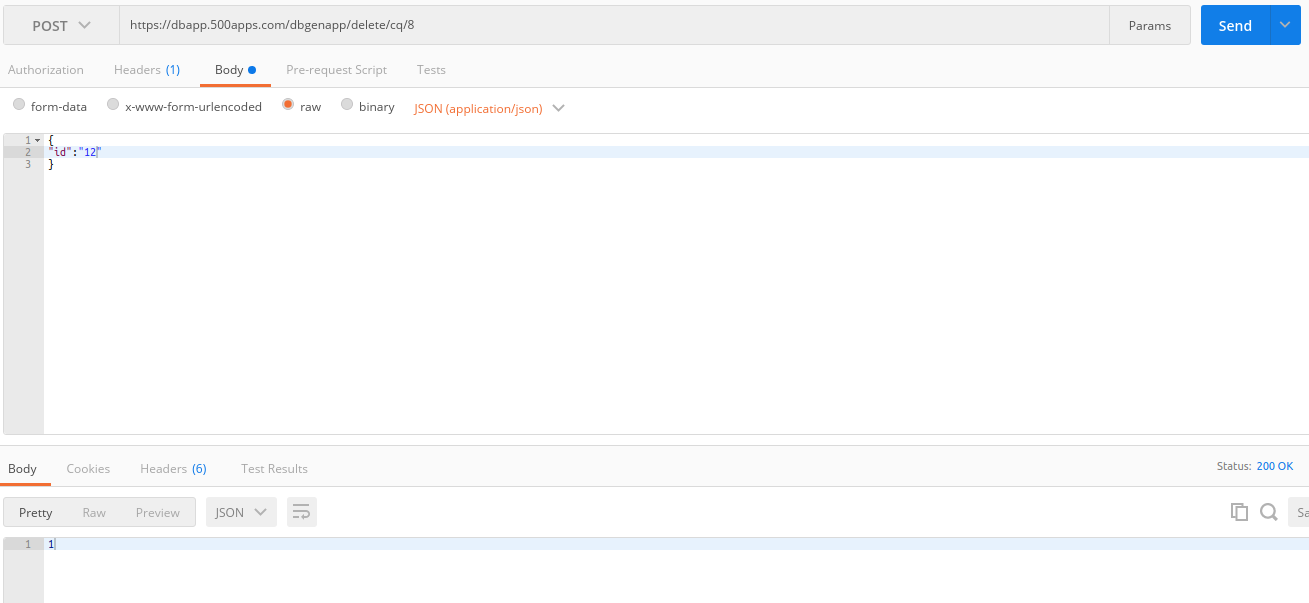
Rest call:https://dbapp.500apps.com/dbgenapp/update/cq/7



TriggerExpression: /delete/cq/{custom.query.value}

Method: **post**

Rest call:https://dbapp.500apps.com/dbgenapp/delete/cq/8



# Regular Expressions

REGEXP performs a pattern match of a string expression against a pattern.

REGEXP is the operator used when performing regular expression pattern matches.

Trigger Expression: /filters

Method: **post**

The input data has to be sent as part of the request body as JSON format. A sample is given below. Any attribute that is not needed for a particular operation should not be included.

The pattern which is to be matched has to be sent with key “regex”.

If we give filters/jsonsearch array then we have to pass the required operator(and/or) in the filter\_condition key.Otherwise, it is to be passed empty.

Data Model:

{

"data": {

"fields": "t.id,t.subject,t.requester\_email",

"tables": [

{

"ticket": "t"

},

{

"ticket\_category": "tc"

}

],

"join": [{

"join\_type": "join",

"table1": "t",

"join\_column1": "status",

"table2": "tc",

"join\_column2": "id"

}],

"filter\_condition":” '',

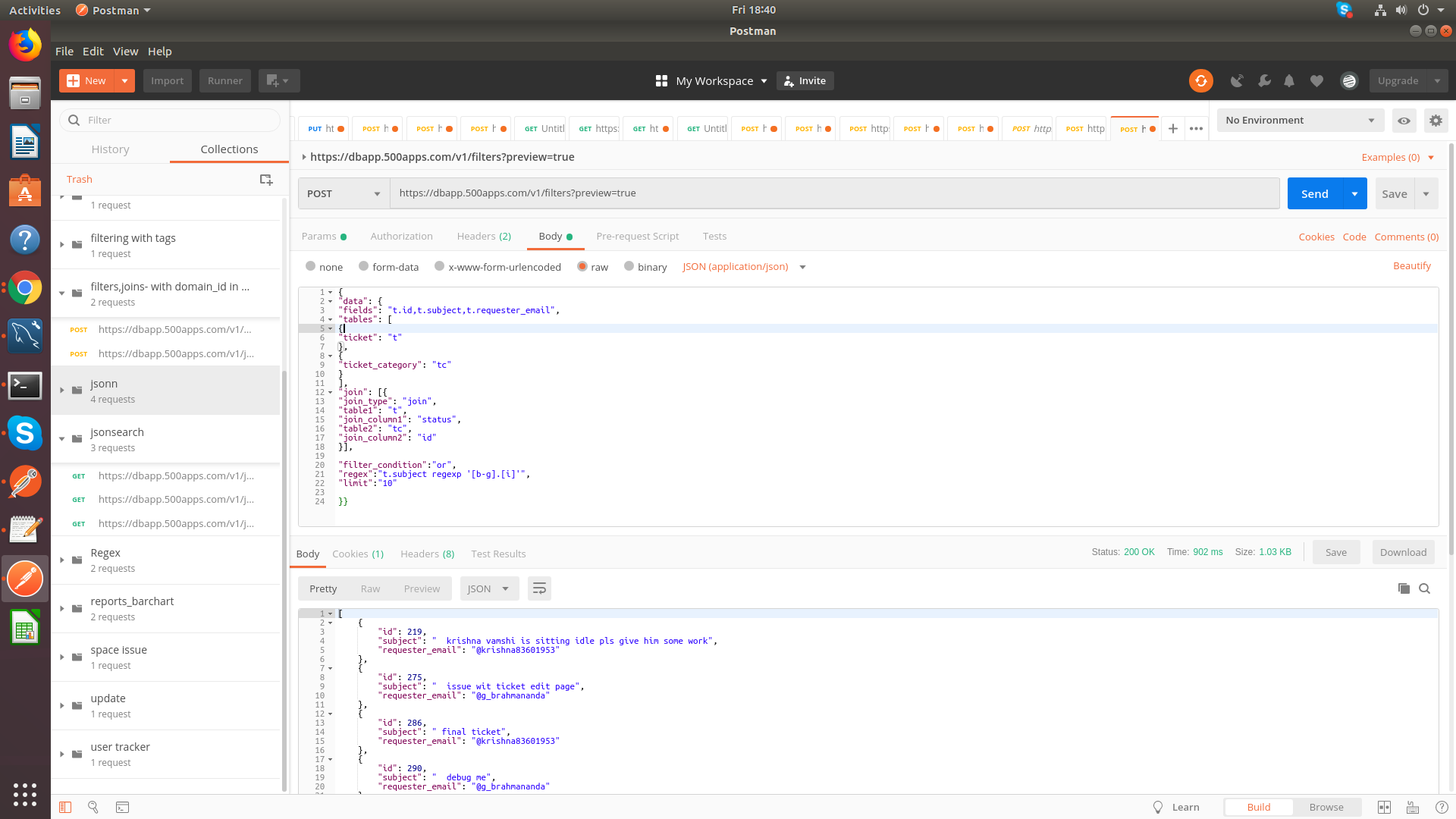
"regex":"t.subject regexp '[b-g].[i]'",

"limit":"10"

}}

Rest call:

<https://dbapp.500apps.com/v1/filters>



# Multi Table Delete

This is to be used when we want to delete data from more than one table which are having a related column between them just like parent table-child table relation.

The Data model is same as joins.

Response would be the number of records deleted from the given tables.

Trigger Expression: /multid

Method: POST

Below is the example Data Model.

Body:

{"data":{

"tables":[

{"message":"m"},

{"resource":"r"},

{"mention":"mn"},

{"like\_message":"l"},

{"star\_message":"s"}

],

"join":[

{

"join\_type":"left join",

"table1":"m",

"join\_column1":"id",

"table2":"r",

"join\_column2":"message\_id"

},

{

"join\_type":"left join",

"table1":"m",

"join\_column1":"id",

"table2":"mn",

"join\_column2":"message\_id"

},

{

"join\_type":"left join",

"table1":"m",

"join\_column1":"id",

"table2":"l",

"join\_column2":"message\_id"

},

{

"join\_type":"left join",

"table1":"m",

"join\_column1":"id",

"table2":"s",

"join\_column2":"message\_id"

}

],

"where":"(m.id=8145 or m.thread\_id=8145 or parent\_message\_id=8145)"

}}

Rest call: https://dbapp.500apps.com/v1/multid

