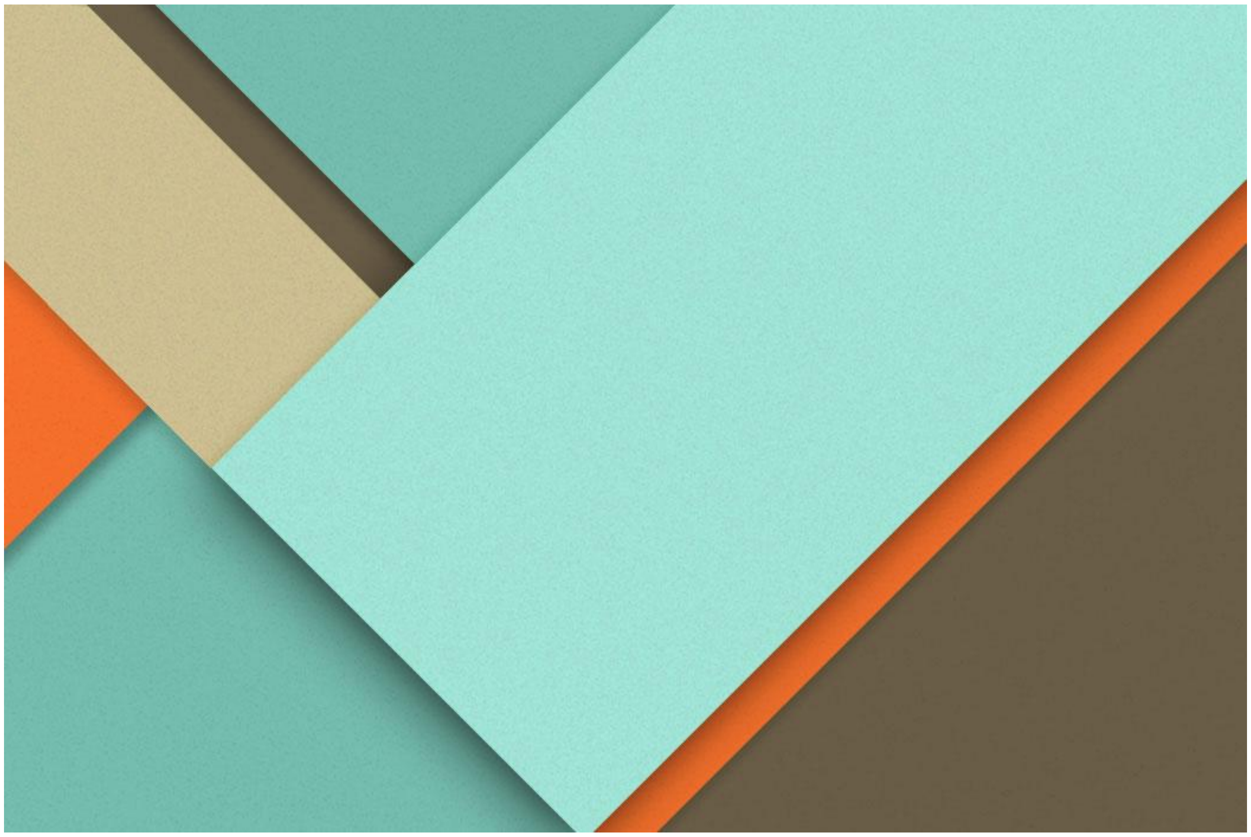


Bangdb:

Command line interface

INDEX

INTRODUCTION

BangDB Command Line Interface (CLI) allows users to interact with BangDB in an easy and efficient manner. Users may perform almost all tasks using the cli. The cli could be used to do several different things in different areas as shown below.

1. Database
2. Stream
3. BRS - BangDB Resource Server
4. Machine learning
5. Replication
6. Agent

While working with CLI there are few points to keep in mind

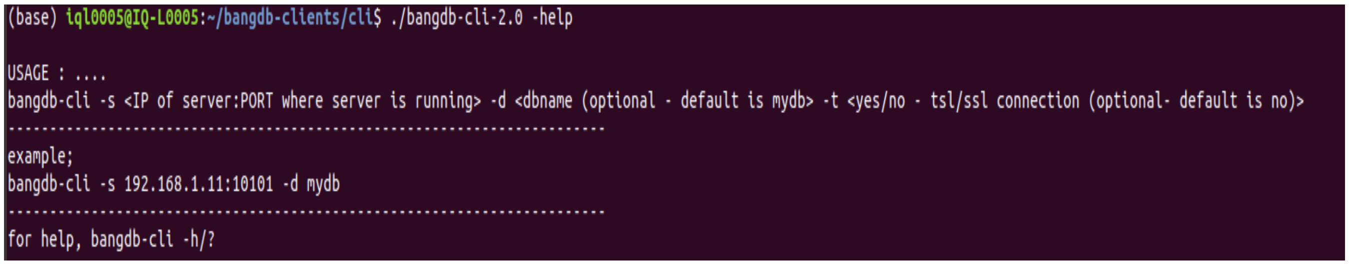
1. CLI reads one line at a time and hence we must complete the command in a single line. CLI interprets the command once users press enter.
2. "_pk" is used for primary keys in the queries
3. Bangdb CLI doesn't need upper case for reserved names or commands
4. There are several commands which creates workflow to complete the process such as schema or app creation, table creating, training model etc

Getting Started with BangDB CLI

Requirements: To run CLI, the bangdb.config file is required. So users should place the bangdb.config file in the same directory as CLI

CLI help command :

From the CLI folder open the terminal console then enter command "**./bangdb-cli-2.0 -help**". This will list all options that can be used to set or connect cli to server.



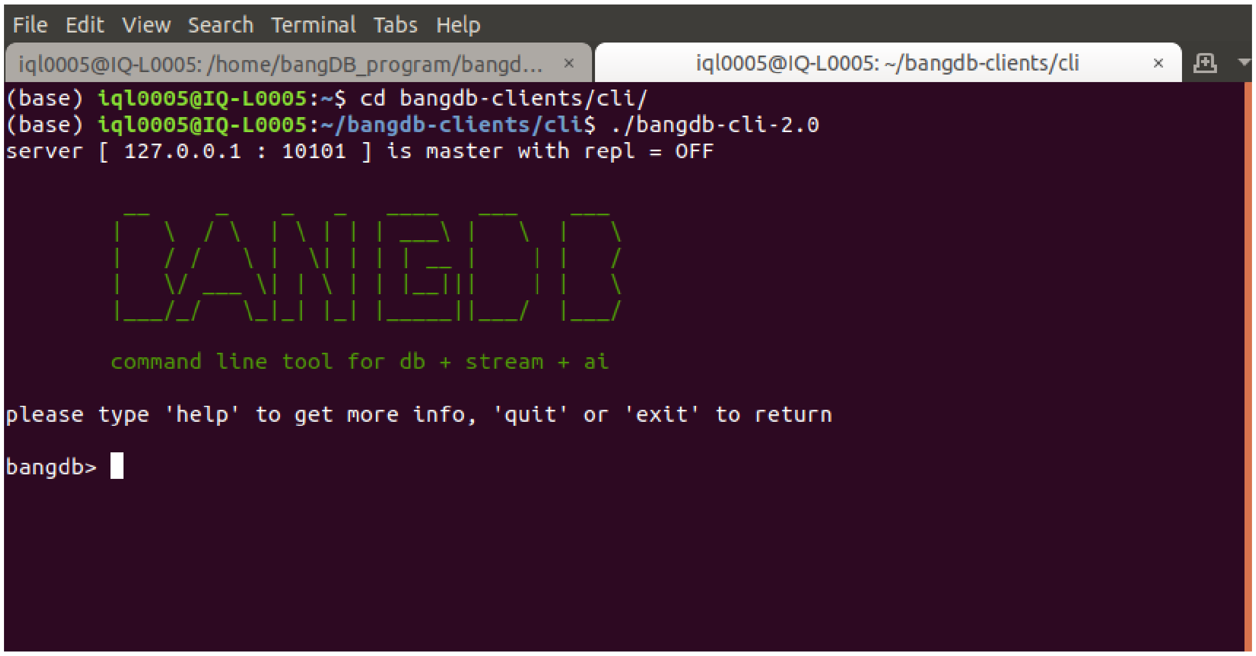
```
(base) iql0005@IQ-L0005:~/bangdb-clients/cli$ ./bangdb-cli-2.0 -help

USAGE : ....
bangdb-cli -s <IP of server:PORT where server is running> -d <dbname (optional - default is mydb> -t <yes/no - tsl/ssl connection (optional- default is no)>
---------------------------------------------------------------------
example;
bangdb-cli -s 192.168.1.11:10101 -d mydb
---------------------------------------------------------------------
for help, bangdb-cli -h/?
```

1. Suppose there are two server running one at port 10101 and other at port 10102 and we want to connect to port 10102 then the command the start CLI should be

   " **./bangdb-cli-2.0 -s 0.0.0.0:10102**"
2. If the database name is Data1 and we want to connect to it then the command is "**./bangdb-cli-2.0 -s 0.0.0.0:10102 -d Data1**"

Starting CLI:

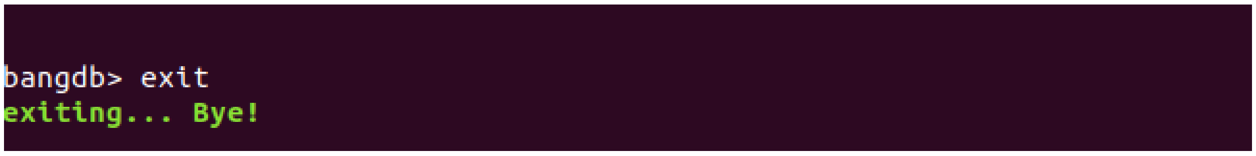
Starting CLi with command "**./bangdb-cli-2.0"** as the server is running at port 0.0.0.0:10101 and database is mydb (default database)


File Edit View Search Terminal Tabs Help
iql0005@IQ-L0005: /home/bangDB_program/bangd...
iql0005@IQ-L0005: ~/bangdb-clients/cli

```
(base) iql0005@IQ-L0005:~$ cd bangdb-clients/cli/
(base) iql0005@IQ-L0005:~/bangdb-clients/cli$ ./bangdb-cli-2.0
server [ 127.0.0.1 : 10101 ] is master with repl = OFF

        command line tool for db + stream + ai

please type 'help' to get more info, 'quit' or 'exit' to return

bangdb>
```

Stopping or Exit CLI:

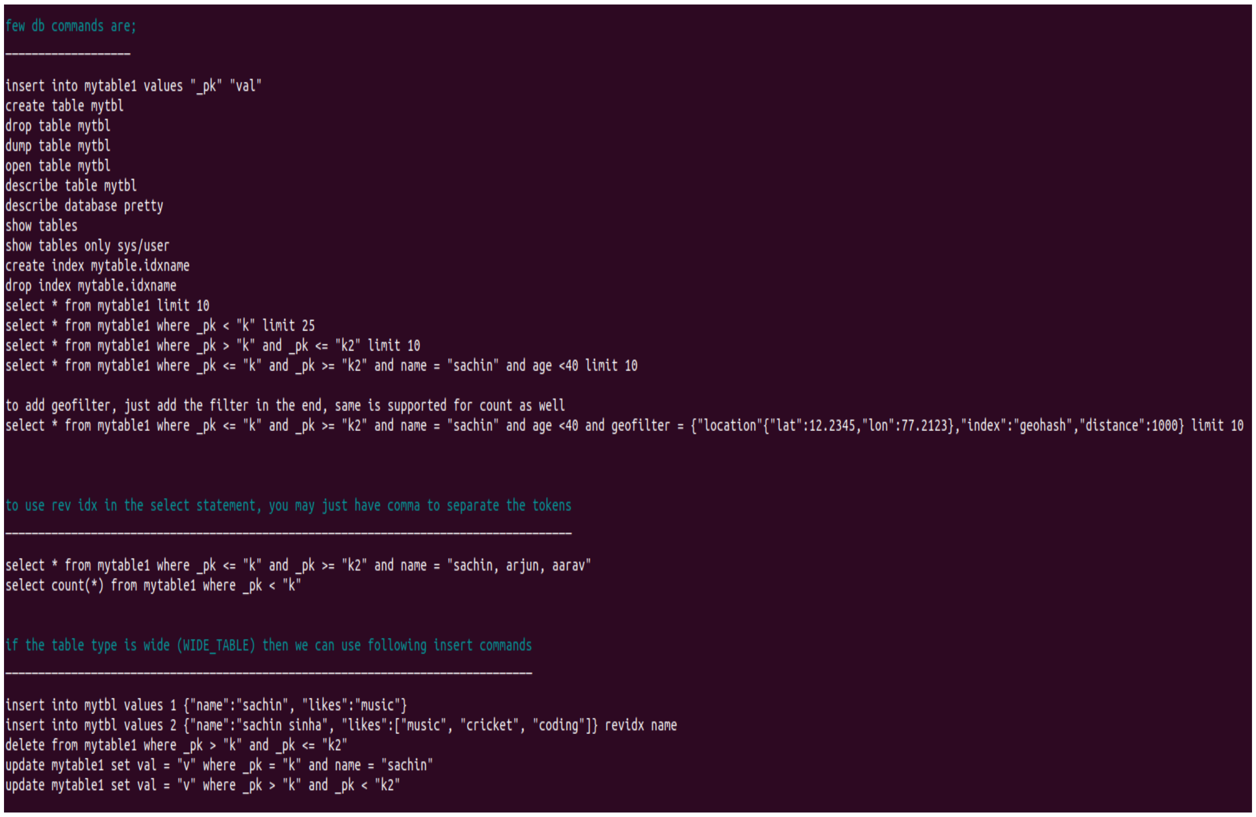
To stop CLI, Users can enter "**exit**" or can press **clt+C**



```
bangdb> exit
exiting... Bye!
```

BangDB Database:

On the CLI console, the users can type "**help db**" to get all the database related commands. The result of help db is shown below. Followed by an example and commands that users can try while going through this document.



```
few db commands are;
-------------------

insert into mytable1 values "_pk" "val"
create table mytbl
drop table mytbl
dump table mytbl
open table mytbl
describe table mytbl
describe database pretty
show tables
show tables only sys/user
create index mytable.idxname
drop index mytable.idxname
select * from mytable1 limit 10
select * from mytable1 where _pk < "k" limit 25
select * from mytable1 where _pk > "k" and _pk <= "k2" limit 10
select * from mytable1 where _pk <= "k" and _pk >= "k2" and name = "sachin" and age <40 limit 10

to add geofilter, just add the filter in the end, same is supported for count as well
select * from mytable1 where _pk <= "k" and _pk >= "k2" and name = "sachin" and age <40 and geofilter = {"location"{"lat":12.2345,"lon":77.2123},"index":"geohash","distance":1000} limit 10



to use rev idx in the select statement, you may just have comma to separate the tokens
-------------------------------------------------------------------------------------

select * from mytable1 where _pk <= "k" and _pk >= "k2" and name = "sachin, arjun, aarav"
select count(*) from mytable1 where _pk < "k"


if the table type is wide (WIDE_TABLE) then we can use following insert commands
-------------------------------------------------------------------------------

insert into mytbl values 1 {"name":"sachin", "likes":"music"}
insert into mytbl values 2 {"name":"sachin sinha", "likes":["music", "cricket", "coding"]} revidx name
delete from mytable1 where _pk > "k" and _pk <= "k2"
update mytable1 set val = "v" where _pk = "k" and name = "sachin"
update mytable1 set val = "v" where _pk > "k" and _pk < "k2"
```

We will be using an example to explain the steps and commands related to Database for better understanding.

- Creating table

Let us create a table by the name CUSTOMERS having fields:

slno	Field	Field_name	Data type
1	Unique id for a customer	CustID	string
2	Customer name	Name	string
3	Gender of the customer	gender	String
4	Age of the customer	age	double

5	Customer Address	address	string
6	Customer Salary	salary	double

To create a table, Users have to enter the command “create table table_name”.

Here, we have to Enter “**create table CUSTOMERS**” --- this will start a workflow for table creation.

1. Table Type [NORMAL_TABLE, KV (0) | WIDE_TABLE, Documents (1) | LARGE_TABLE, large objects/files (2) | PREMITIVE_TABLE, like column (3)] (or Enter for default (0)): 1
   [Here, select wide table]

2. Is it a SW(Sliding Window) table? y/n (or Enter for default (n)): n
   [Selecting n (no) as we don’t want Sliding Window]

3. allow reverse index as well? y/n (or Enter for default (n)): n
   [Selecting No for reverse indexing]

4. allow duplicate primary keys as well? y/n(or Enter for default (n)): n
   [We don’t want any duplicate data]

5. Key type [NORMAL_KEY(string type) (0) | COMPOSITE_KEY(also string type) (1) | NORMAL_KEY_LONG (long type) (2)] (or Enter for default (0)): 2
   [Selecting integer as the data type for primary index]

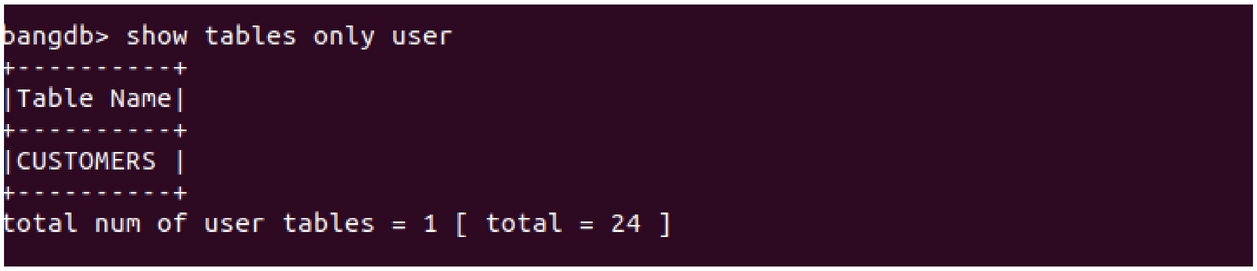
6. primary key arrangement (index) type [BTREE (2) | EXTHASH (1)] (or Enter for default (2)):
7. Method for key sort [Lexicographically (1) | Quasi Lexicographically (2)] (or Enter for default (2)):
   [above two are method used for storing and retrieving data]

8. Direction for key sort [Ascending (3) | Descending (4)] (or Enter for default (3)):
   [For this example user can select default]

With the above steps CUSTOMERS table is created. In bangDB, we don’t have to separately specify the attributes or their data type.

- To list user created tables

The command **“show tables only user**” will list all the user created table presents in the system.



```
bangdb> show tables only user
+----------+
|Table Name|
+----------+
|CUSTOMERS |
+----------+
total num of user tables = 1 [ total = 24 ]
```

- Inserting data into the table

Now let's insert some data into the above table from CLI.

Command for inserting data “**insert into** table_name **values** index_num { data in json}”

Inserting following data from CLI (one at a time. Users can copy the below lines and paste it on the CLI console.)

```
insert into CUSTOMERS values 1
{"CustID":"123","name":"Anurag","gender":"M","age":24,"salary":56000}

insert into CUSTOMERS values 2
{"CustID":"983","name":"Aurn","gender":"M","age":24,"salary":36000}

insert into CUSTOMERS values 3
{"CustID":"443","name":"Vignesh","gender":"M","age":21,"salary":26000}

insert into CUSTOMERS values 4
{"CustID":"809","name":"Pooja","gender":"F","age":28,"salary":76000}

insert into CUSTOMERS values 5
{"CustID":"223","name":"Varun","gender":"M","age":24,"salary":56000}

insert into CUSTOMERS values 6
{"CustID":"383","name":"prabhu","gender":"M","age":24,"salary":36000}

insert into CUSTOMERS values 7
{"CustID":"563","name":"Ajja","gender":"M","age":21,"salary":26000}

insert into CUSTOMERS values 8
{"CustID":"318","name":"Neetu","gender":"F","age":28,"salary":76000}
```

- To create index

Let's make CustID as an index. To do this user have to enter “**create index CUSTOMERS.CustID**”---- this will start a workflow

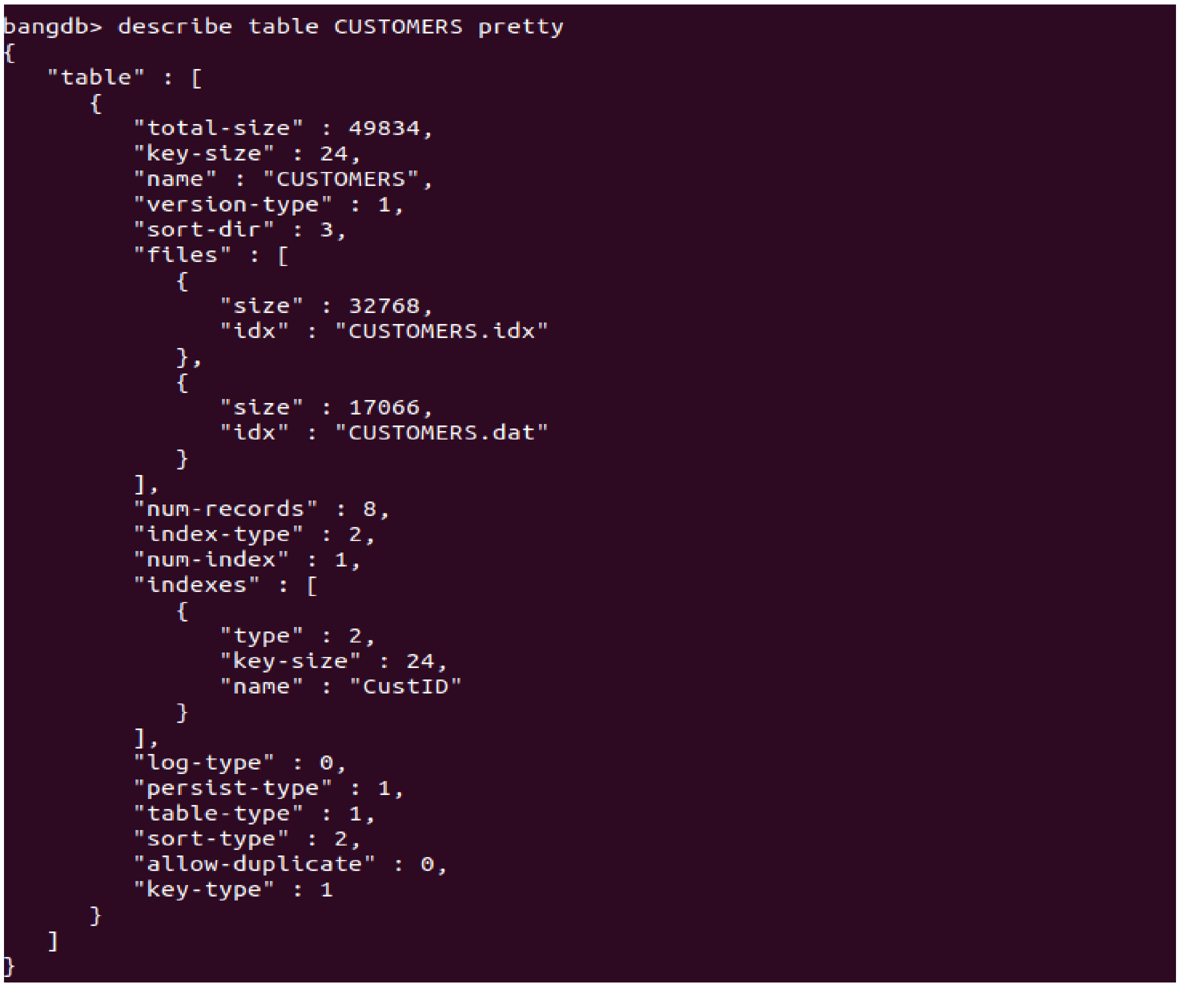
1. Key Type [NORMAL_KEY (1) | NORMAL_KEY_LONG (2) | COMPOSITE_KEY (3)] (or Enter to set default (1)): 3
   [As CustID is string, we will select key type (data type) as Composite]

2. Key size (or Enter to set default (24)):
   [setting the size, we are setting key size as 24]

3. Sort direction [SORT_ASCENDING(3) | SORT_DESCENDING(4)] (or Enter to set default (3)):
4. Sort method [LEXICOGRAPH(1) | QUASI_LEXICOGRAPH(2)] (or Enter to set default (2)):
5. Allow duplicate indexes as well? y/n: (or Enter for default (n))
   [For above three, users can enter and default values will be selected]

- To get the description of the CUSTOMERS table

To check the properties of the CUSTOMERS table "**describe table CUSTOMERS pretty**". This command will display the properties of above table like shown below



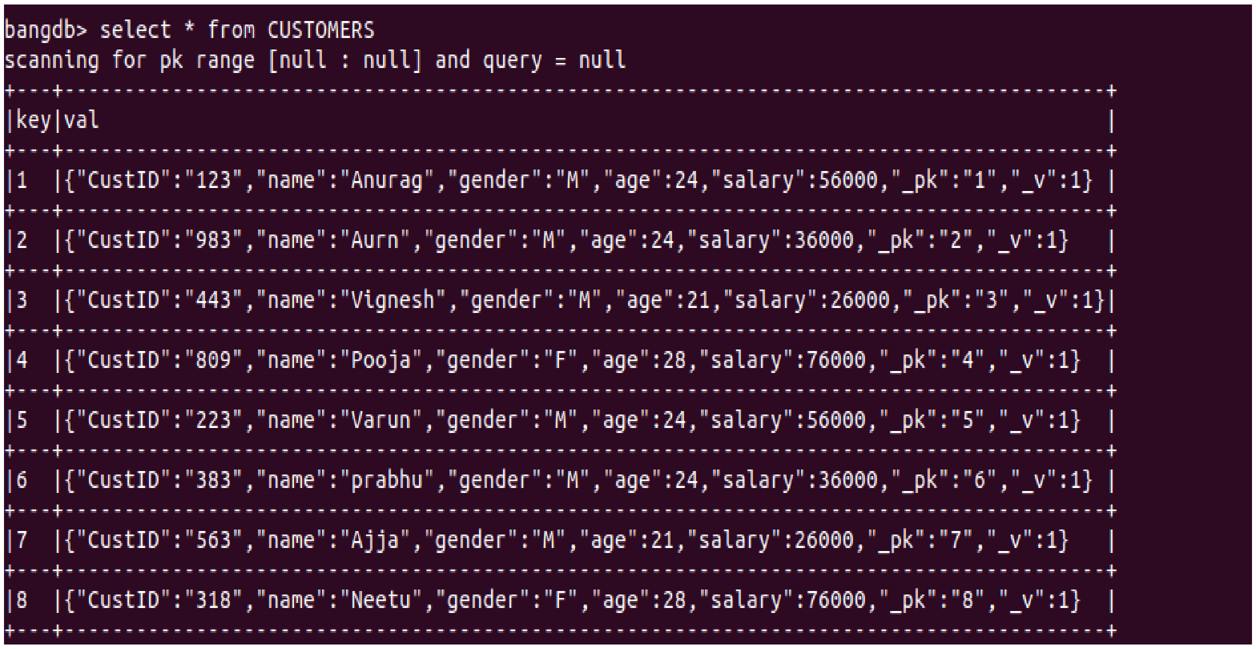
```
bangdb> describe table CUSTOMERS pretty
{
   "table" : [
      {
         "total-size" : 49834,
         "key-size" : 24,
         "name" : "CUSTOMERS",
         "version-type" : 1,
         "sort-dir" : 3,
         "files" : [
            {
               "size" : 32768,
               "idx" : "CUSTOMERS.idx"
            },
            {
               "size" : 17066,
               "idx" : "CUSTOMERS.dat"
            }
         ],
         "num-records" : 8,
         "index-type" : 2,
         "num-index" : 1,
         "indexes" : [
            {
               "type" : 2,
               "key-size" : 24,
               "name" : "CustID"
            }
         ],
         "log-type" : 0,
         "persist-type" : 1,
         "table-type" : 1,
         "sort-type" : 2,
         "allow-duplicate" : 0,
         "key-type" : 1
      }
   ]
}
```

- To drop the index

Let’s drop or remove Custid as an index. To this users have to enter “**drop index CUSTOMERS.CustID**”. Now if the users perform describe table command, we see that the indexes section will not be present.

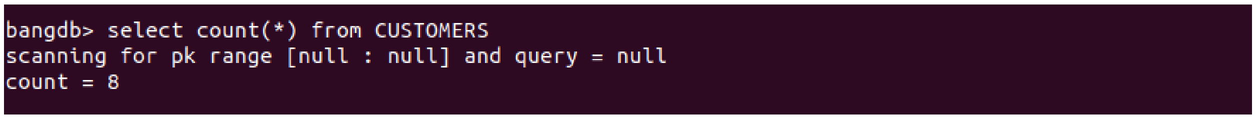
- Retrieving data based on conditions

Command to retrieve all data “**select * from CUSTOMERS**”. This will display all events in the table on the console



```
bangdb> select * from CUSTOMERS
scanning for pk range [null : null] and query = null
+---+--------------------------------------------------------------------------------+
|key|val                                                                             |
+---+--------------------------------------------------------------------------------+
|1  |{"CustID":"123","name":"Anurag","gender":"M","age":24,"salary":56000,"_pk":"1","_v":1} |
+---+--------------------------------------------------------------------------------+
|2  |{"CustID":"983","name":"Aurn","gender":"M","age":24,"salary":36000,"_pk":"2","_v":1}   |
+---+--------------------------------------------------------------------------------+
|3  |{"CustID":"443","name":"Vignesh","gender":"M","age":21,"salary":26000,"_pk":"3","_v":1}|
+---+--------------------------------------------------------------------------------+
|4  |{"CustID":"809","name":"Pooja","gender":"F","age":28,"salary":76000,"_pk":"4","_v":1}  |
+---+--------------------------------------------------------------------------------+
|5  |{"CustID":"223","name":"Varun","gender":"M","age":24,"salary":56000,"_pk":"5","_v":1}  |
+---+--------------------------------------------------------------------------------+
|6  |{"CustID":"383","name":"prabhu","gender":"M","age":24,"salary":36000,"_pk":"6","_v":1} |
+---+--------------------------------------------------------------------------------+
|7  |{"CustID":"563","name":"Ajja","gender":"M","age":21,"salary":26000,"_pk":"7","_v":1}   |
+---+--------------------------------------------------------------------------------+
|8  |{"CustID":"318","name":"Neetu","gender":"F","age":28,"salary":76000,"_pk":"8","_v":1}  |
+---+--------------------------------------------------------------------------------+
```

And to get the total number of events “select count(*) from CUSTOMERS”

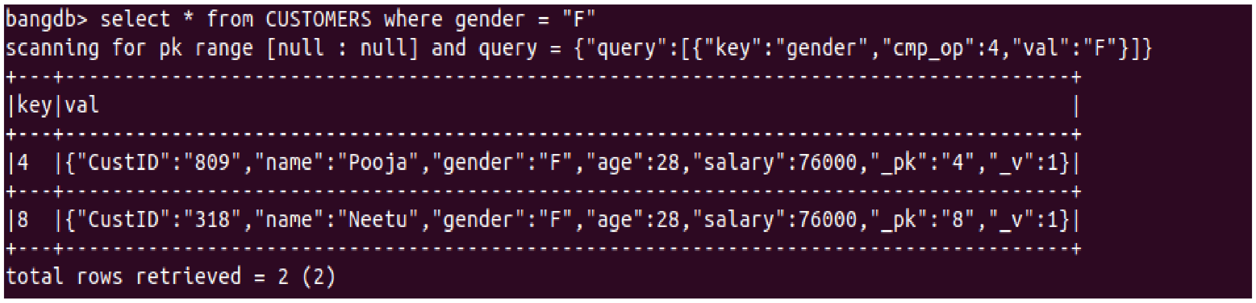


```
bangdb> select count(*) from CUSTOMERS
scanning for pk range [null : null] and query = null
count = 8
```

Let's try a few conditions to retrieve data based on these conditions.

1. To retrieve data for only Females

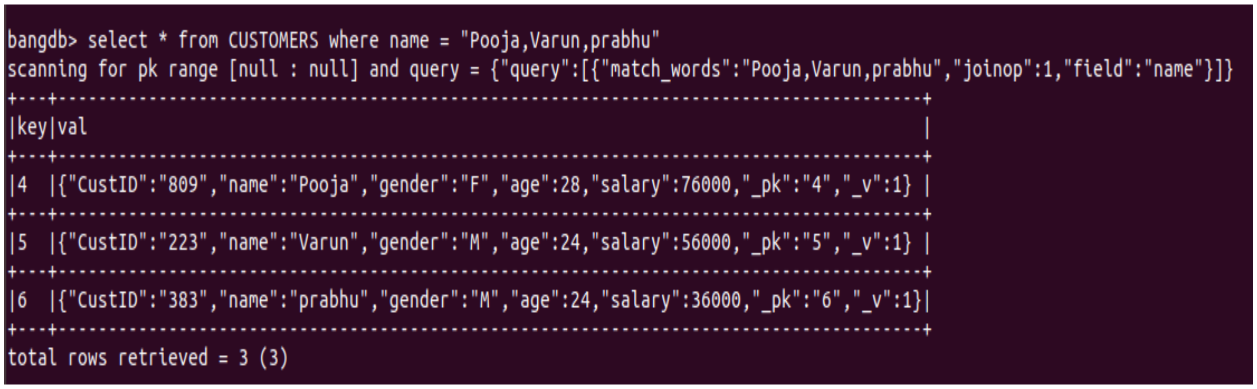
   The query is : **select * from CUSTOMERS where gender = "F"**



```
bangdb> select * from CUSTOMERS where gender = "F"
scanning for pk range [null : null] and query = {"query":[{"key":"gender","cmp_op":4,"val":"F"}]}
+---+--------------------------------------------------------------------------------+
|key|val                                                                             |
+---+--------------------------------------------------------------------------------+
|4  |{"CustID":"809","name":"Pooja","gender":"F","age":28,"salary":76000,"_pk":"4","_v":1}|
+---+--------------------------------------------------------------------------------+
|8  |{"CustID":"318","name":"Neetu","gender":"F","age":28,"salary":76000,"_pk":"8","_v":1}|
+---+--------------------------------------------------------------------------------+
total rows retrieved = 2 (2)
```

2. To retrieve data for where name = Pooja or Varun or pradhu

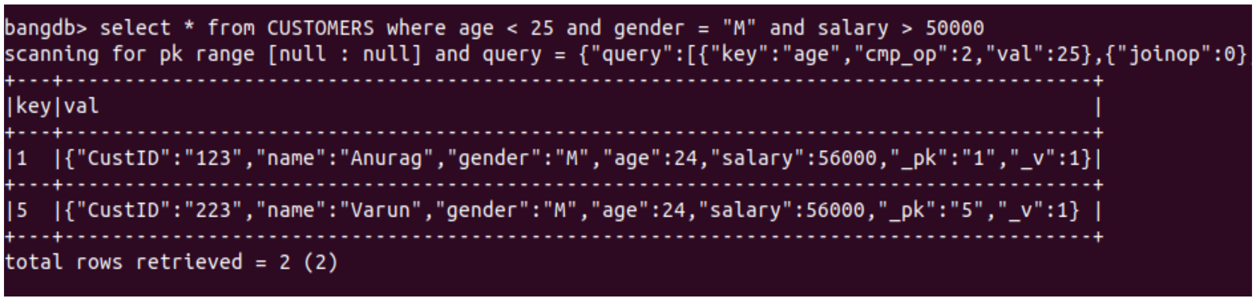
   The query is : **select * from CUSTOMERS where name = "Pooja,Varun,prabhu"**



```
bangdb> select * from CUSTOMERS where name = "Pooja,Varun,prabhu"
scanning for pk range [null : null] and query = {"query":[{"match_words":"Pooja,Varun,prabhu","joinop":1,"field":"name"}]}
+---+-----------------------------------------------------------------------------+
|key|val                                                                          |
+---+-----------------------------------------------------------------------------+
|4  |{"CustID":"809","name":"Pooja","gender":"F","age":28,"salary":76000,"_pk":"4","_v":1} |
+---+-----------------------------------------------------------------------------+
|5  |{"CustID":"223","name":"Varun","gender":"M","age":24,"salary":56000,"_pk":"5","_v":1} |
+---+-----------------------------------------------------------------------------+
|6  |{"CustID":"383","name":"prabhu","gender":"M","age":24,"salary":36000,"_pk":"6","_v":1}|
+---+-----------------------------------------------------------------------------+
total rows retrieved = 3 (3)
```

3. To retrieve data where age is less than 25 and gender is male and salary is above 50000.

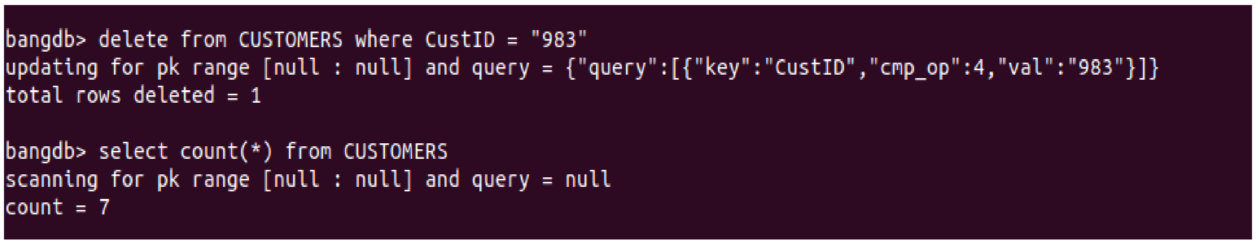
   The query is: **select * from CUSTOMERS where age < 25 and gender = "M" and salary > 50000**



```
bangdb> select * from CUSTOMERS where age < 25 and gender = "M" and salary > 50000
scanning for pk range [null : null] and query = {"query":[{"key":"age","cmp_op":2,"val":25},{"joinop":0}
+---+-----------------------------------------------------------------------------+
|key|val                                                                          |
+---+-----------------------------------------------------------------------------+
|1  |{"CustID":"123","name":"Anurag","gender":"M","age":24,"salary":56000,"_pk":"1","_v":1}|
+---+-----------------------------------------------------------------------------+
|5  |{"CustID":"223","name":"Varun","gender":"M","age":24,"salary":56000,"_pk":"5","_v":1} |
+---+-----------------------------------------------------------------------------+
total rows retrieved = 2 (2)
```

- To delete a wrong Record from table

Let's delete the record where the CustID is 983. To do this the query is "**delete from CUSTOMERS where CustID = "983"**



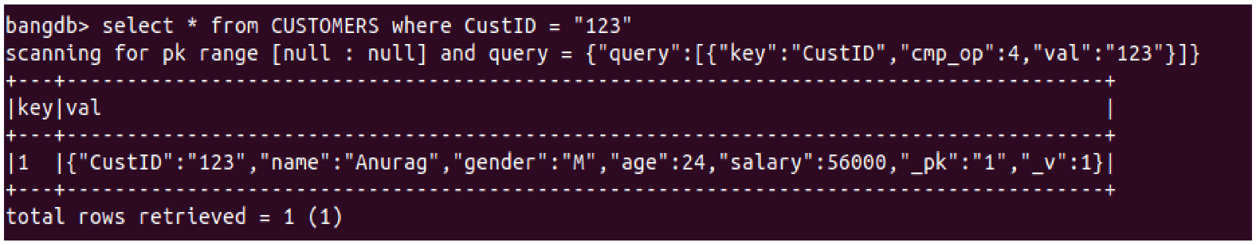
```
bangdb> delete from CUSTOMERS where CustID = "983"
updating for pk range [null : null] and query = {"query":[{"key":"CustID","cmp_op":4,"val":"983"}]}
total rows deleted = 1

bangdb> select count(*) from CUSTOMERS
scanning for pk range [null : null] and query = null
count = 7
```

We have deleted 1 event and the check we are counting the total events in the table was 8 before now its 7.

- To update or change a record

Suppose we want to correct a record which is wrong. Let's say the record with CustID 123, the name and gender is incorrect.

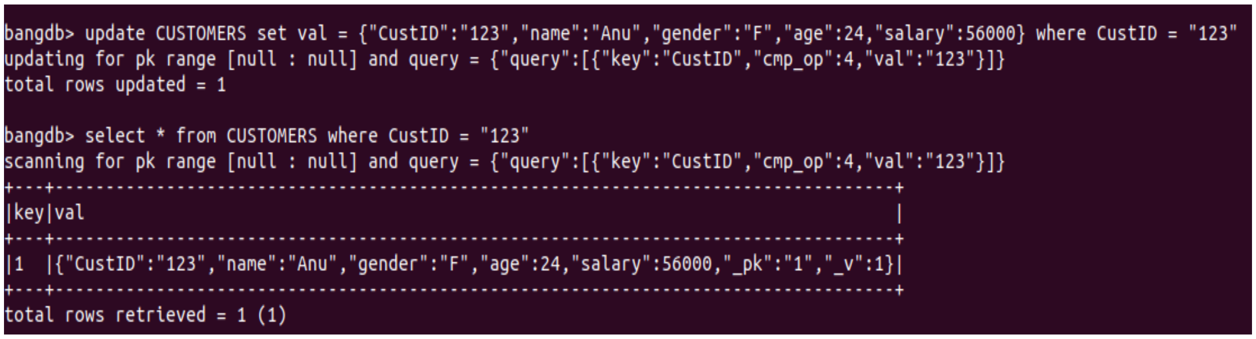
Now the value is :



```
bangdb> select * from CUSTOMERS where CustID = "123"
scanning for pk range [null : null] and query = {"query":[{"key":"CustID","cmp_op":4,"val":"123"}]}
+---+--------------------------------------------------------------------------------+
|key|val                                                                             |
+---+--------------------------------------------------------------------------------+
|1  |{"CustID":"123","name":"Anurag","gender":"M","age":24,"salary":56000,"_pk":"1","_v":1}|
+---+--------------------------------------------------------------------------------+
total rows retrieved = 1 (1)
```

We want to change name to “Anu” and gender to “F” for CustID = 123

To do this the query is “**update CUSTOMERS set val = {"CustID":"123","name":"Anu","gender":"F","age":24,"salary":56000} where CustID = "123"**

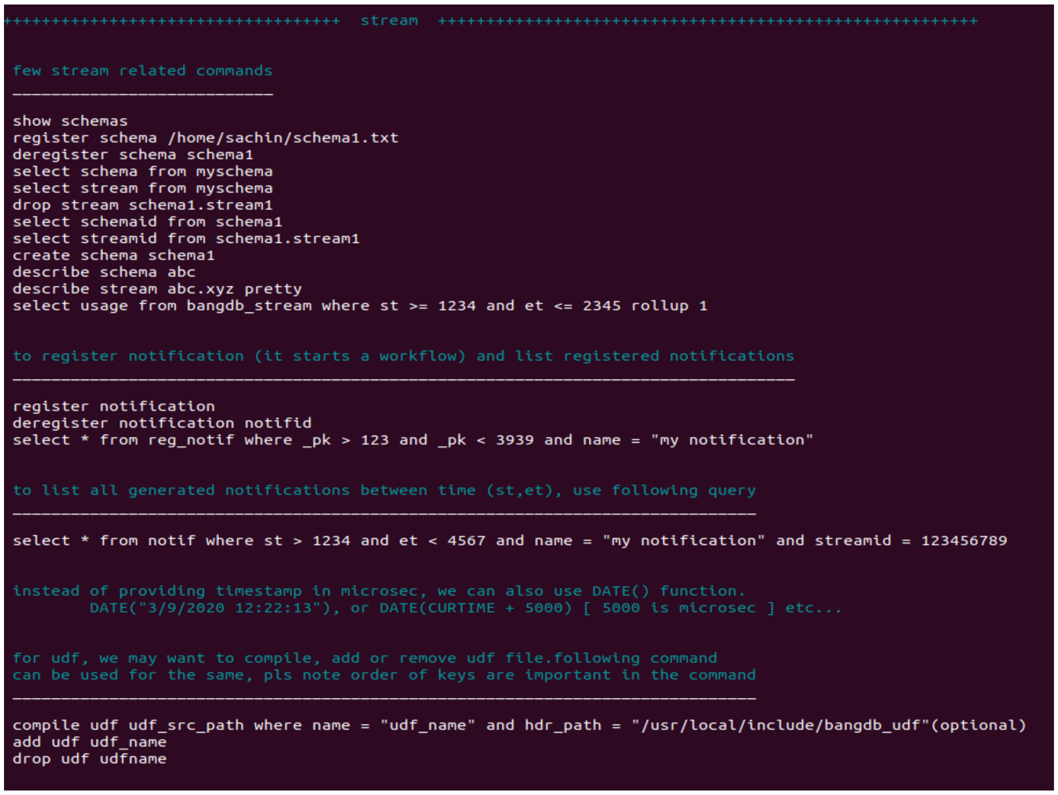


```
bangdb> update CUSTOMERS set val = {"CustID":"123","name":"Anu","gender":"F","age":24,"salary":56000} where CustID = "123"
updating for pk range [null : null] and query = {"query":[{"key":"CustID","cmp_op":4,"val":"123"}]}
total rows updated = 1

bangdb> select * from CUSTOMERS where CustID = "123"
scanning for pk range [null : null] and query = {"query":[{"key":"CustID","cmp_op":4,"val":"123"}]}
+---+--------------------------------------------------------------------------------+
|key|val                                                                             |
+---+--------------------------------------------------------------------------------+
|1  |{"CustID":"123","name":"Anu","gender":"F","age":24,"salary":56000,"_pk":"1","_v":1}|
+---+--------------------------------------------------------------------------------+
total rows retrieved = 1 (1)
```

We can check by performing “**select * from CUSTOMERS where CustID = "123"** as shown above. The name and gender got replaced.

Schema and stream

On entering the command "**help stream**", We get a list of all the commands related to schema and stream as shown in the figure below. We will be using an example to explain different features and Users can also try this as then go through the document.



```
+++++++++++++++++++++++++++++++++++  stream  ++++++++++++++++++++++++++++++++++++++++++++++++++++++++

 few stream related commands
 ___________________________

 show schemas
 register schema /home/sachin/schema1.txt
 deregister schema schema1
 select schema from myschema
 select stream from myschema
 drop stream schema1.stream1
 select schemaid from schema1
 select streamid from schema1.stream1
 create schema schema1
 describe schema abc
 describe stream abc.xyz pretty
 select usage from bangdb_stream where st >= 1234 and et <= 2345 rollup 1

 to register notification (it starts a workflow) and list registered notifications
 _________________________________________________________________________________

 register notification
 deregister notification notifid
 select * from reg_notif where _pk > 123 and _pk < 3939 and name = "my notification"

 to list all generated notifications between time (st,et), use following query
 _____________________________________________________________________________

 select * from notif where st > 1234 and et < 4567 and name = "my notification" and streamid = 123456789

 instead of providing timestamp in microsec, we can also use DATE() function.
         DATE("3/9/2020 12:22:13"), or DATE(CURTIME + 5000) [ 5000 is microsec ] etc...

 for udf, we may want to compile, add or remove udf file.following command
 can be used for the same, pls note order of keys are important in the command
 _____________________________________________________________________________

 compile udf udf_src_path where name = "udf_name" and hdr_path = "/usr/local/include/bangdb_udf"(optional)
 add udf udf_name
 drop udf udfname
```

Let's consider that we will be getting stream data for customer when they purchase a product and the attributes or fields are:

slno	Field	Field_name	Data type
1	Unique id for a customer	CustID	string
2	Product name	productID	string
3	Quantity	qty	long
4	Product cost	cost	double

1. Creating Schema to collect data

We will be creating a schema or app which contains all streams related to this example. Here, we will create a schema name “sch” with one stream name customers. Enter command **“create schema sch**” this will start a workflow

1. **create schema sch**
   [sch is the name of the schema for this use-case. Its user-defined]

   For here on the workflow will start

2. **Do you wish to read earlier saved schema for editing/adding? [yes | no]: no**
   [Here, we can upload a schema from local system]

3. **what's the name of the stream that you wish to add?: customers**
   [Name of the stream, we can only create one stream at a time]

4. **what's the type of the stream [raw/normal primary (1) | fltr output (2) | join/cep output (3) | Entity (4) | skip (5)] (or Enter for default (1)): 1**
   [Here we have to define whether this stream is raw stream (uploading data directly) or its a stream resulting from filter, join or cep operations]

5. **what's the size of the sliding window in seconds: 86400**
   [Here we have to define sliding window]

6. **What would you like to add (press Enter when done) [attr (1) | catr (2) | refr (3) | gpby (4) | fltr (5) | join (6) | entity (7) | cep (8) | notifs (9)]: 1**
   [Here, we define what we are planning to add to the stream, first we have to add attributes/columns one by one then we can add group-by or join or etc...]

7. **add attributes...**

   A. **attribute name:** CustID
      [We have to add 4 attribute but one attribute at a time]

   B. **attribute type (press Enter for default(5)) [string(5) | long(9) | double (11)]: 5**
      [defining data type for attributes, we are setting 5 (string type) for CustID attribute.]

   C. **enable sidx [0 | 1]: 1**
      [Enabling secondary index on customer_id. Help in performing search operations]

   D. **attribute key size in bytes (press Enter for default(24 bytes)):** 16
      [setting key size for string attributes]

   E. **enable ridx [0 | 1]: 0**
      [its for enabling reverse indexing]

   F. **enable stat [none(0) | count (1) | unique count (2)]: 1**
      [Here, we define the statistical operation that we want to perform on the attribute. As we have selected the data type as string so we can perform count and unique count

operation.]

8. **add another attribute ? [yes | no]: yes**
   attribute name: qty
   attribute type (press Enter for default(5)) [string(5) | long(9) | double (11)]: 11
   enable stat [none(0) | count (1) | running stat (3)]: 3

9. **add another attribute ? [yes | no]: yes**
   attribute name: productID
   attribute type (press Enter for default(5)) [string(5) | long(9) | double (11)]: 5
   enable sidx [0 | 1]: 0
   enable ridx [0 | 1]: 0
   enable stat [none(0) | count (1) | unique count (2)]: 2

10. **add another attribute ? [yes | no]: yes**
    attribute name: cost
    attribute type (press Enter for default(5)) [string(5) | long(9) | double (11)]: 11
    enable stat [none(0) | count (1) | running stat (3)]: 3

11. **add another attribute ? [yes | no]: no**
    [Once we have entered all the attributes, we have to select no to exit]

What would you like to add (press Enter when done) [attr (1) | catr (2) | refr (3) | gpby (4) | fltr (5) | join (6) | entity (7) | cep (8) | notifs (9)]:

add another stream ? [yes | no]: no

```
Update Schema :
{
  "schema" : "sch",
  "streams" : [
          {
          "type" : 1,
          "attr" : [
          {"stat" : 1,"name" : "CustID","type" : 5},
          {"type" : 11,"name" : "qty","stat" : 3},
          {"stat" : 2,"name" : "productID","type" : 5},
          {"type" : 11,"name" : "cost","stat" : 3}
          ],
          "swsz" : 86400,
          "inpt" : [],
          "name" : "customers"
          }
  ]
}
```

12. **Do you wish to register the schema now? [yes | no]: yes**

We will get message "schema [sch] registered successfully"

- Want to add group- by

Suppose we want to monitor the quantity of products based on productID. To do this we have to add groupby to the schema “sch”. Steps involved in adding groupby is the same as creating schema.

Enter the command “**create schema sml”** then enter the name of the stream to which we want to add the group by.

1. **create schema sch**

[Message displayed on screen schema [sch] already exists in the db, fetching the details for editing]

2. **Do you wish to view the schema before editing/adding? [yes | no]: no**
   [If users select yes, it will display the schema]

3. **What's the name of the stream that you wish to add?: customers**
   [stream name to which will be adding group by]

4. **What would you like to add (press Enter when done) [attr (1) | catr (2) | refr (3) | gpby (4) | fltr (5) | join (6) | entity (7) | cep (8) | notifs (9)]: 4**
   [select option 4- groupby]

add groupby (gpby)...

5. **name of the attribute that would be aggregated: qty**
   [attribute to be grouped]

6. **enter name of groupby attributes (press Enter once done): productID**
   [attribute based on which we want to do group by]

7. **attribute key size in bytes (note gpby is name mangled with aggr and groupby attr names, hence should be properly allocated): 120**
   [set the key size]

10. **granularity for the aggregate (in seconds): 3600**
    [set the duration in seconds]

11. **enable stat (1,2,3) [count (1) | unique count (2) | running stat (3)]: 3**
    [Select stat, we and running stat]

add another gpby ? [yes | no]: no and Save the schema by entering yes.

- To Filter Customers if their purchase cost is more than 5000.

Now we will be separating all customers who have purchased costs above 5000.

1. **create schema sch**

[Message displayed on screen schema [sch] already exists in the db, fetching the details for editing]

2. **Do you wish to view the schema before editing/adding? [yes | no]: no**
   [If users select yes, it will display the schema]

3. **What's the name of the stream that you wish to add?: customers**
   [stream name to which will be adding group by]

4. **What would you like to add (press Enter when done) [attr (1) | catr (2) | refr (3) | gpby (4) | fltr (5) | join (6) | entity (7) | cep (8) | notifs (9)]**: 5

add filter (fltr)...

5. **filter name: flt1**
   [Filter name, its user define]

6. **enter type of fqry [query (1) | udf (2)]: 1**
   [As we will be using condition for attribute]

7. **enter the filter key type [ATTR (1) | computed (3)] : 1**
   [filtering based on attribute therefore 1]

8. **enter the name of the attribute for comparison from input stream: cost**
   [Name of the attribute]

9. **enter value type [attr val,(ex; $a) (1) | computed (3) | string (5) | long (9) | double (11)]: 11**
10. **enter value: 5000**
11. **enter comparison operation [GT | GTE | LT | LTE | EQ | NE] (of Enter for default EQ): GT** [value and condition for filtering   ]

add another query ? [yes | no]: no

12. **enter name of attributes that will get selected from this stream if filter succeeds (press enter once done):** CustID
13. **enter name of attributes that will get selected from this stream if filter succeeds (press enter once done):** productID
14. **enter name of attributes that will get selected from this stream if filter succeeds (press enter once done):** qty
    [Attribute to be sent to filter stream when condition is satisfied]

15. **name of the output stream, to which these filtered events would be sent: purchaseOver5000**
    [Output stream name]

Add stream purchaseOver5000 by saying yes to

add another stream ? [yes | no]: yes

- Inserting events into the stream

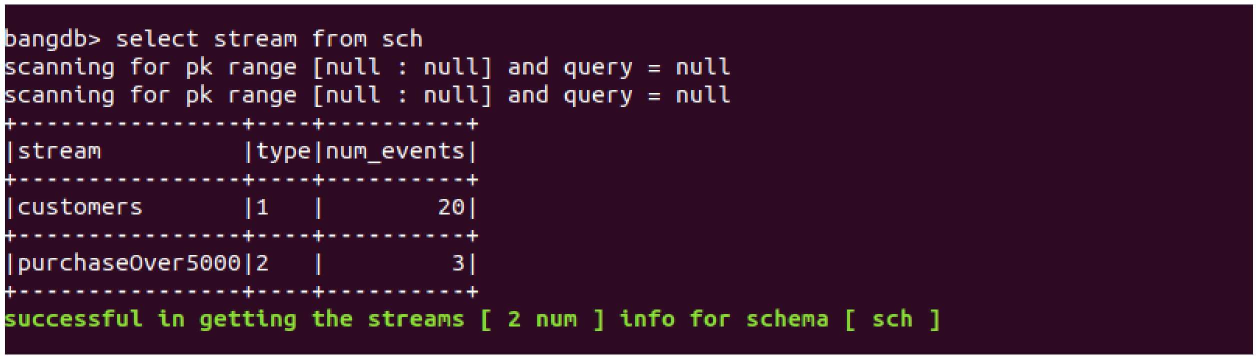
To insert data from CLI the command is "**insert into schema_name.stream_name values null {data in json format}**". Users can use the data below to try it out.

```
insert into sch.customers values null {"CustID":"ad123","productID":"homeUlitcp","qty":3,"cost":849}
insert into sch.customers values null {"CustID":"ad124","productID":"kitUlita","qty":1,"cost":2903}
insert into sch.customers values null {"CustID":"ad163","productID":"kitUlit","qty":1,"cost":149}
insert into sch.customers values null {"CustID":"ad126","productID":"homeUlit","qty":1,"cost":2343}
insert into sch.customers values null {"CustID":"ad127","productID":"kitUlit","qty":2,"cost":674}
insert into sch.customers values null {"CustID":"ad128","productID":"kitUlit","qty":1,"cost":345}
insert into sch.customers values null {"CustID":"ad129","productID":"kitUlit","qty":1,"cost":90}
insert into sch.customers values null {"CustID":"ad133","productID":"kitFanBAJ","qty":1,"cost":6049}
insert into sch.customers values null {"CustID":"ad143","productID":"kitUlit","qty":1,"cost":263}
insert into sch.customers values null {"CustID":"ad153","productID":"EleUlit","qty":5,"cost":1833}
insert into sch.customers values null {"CustID":"ad164","productID":"kitUlit","qty":2,"cost":3692}
insert into sch.customers values null {"CustID":"ad173","productID":"kitUlit","qty":1,"cost":923}
insert into sch.customers values null {"CustID":"ad183","productID":"kitMICROLG","qty":1,"cost":5689}
insert into sch.customers values null {"CustID":"ad193","productID":"EleUlit","qty":1,"cost":222}
insert into sch.customers values null {"CustID":"ad223","productID":"kitUlit","qty":1,"cost":384}
insert into sch.customers values null {"CustID":"ad323","productID":"kitUlit","qty":2,"cost":803}
insert into sch.customers values null {"CustID":"ad499","productID":"kitFriWP","qty":1,"cost":8419}
insert into sch.customers values null {"CustID":"ad523","productID":"EleUlit","qty":6,"cost":1903}
insert into sch.customers values null {"CustID":"ad623","productID":"kitUlit","qty":2,"cost":4902}
insert into sch.customers values null {"CustID":"ad723","productID":"kitUlit","qty":7,"cost":2039}
```

In the above data, we have 3 events where the cost is more than 5000. Now will be performing queries, the chenk stat and events.

- Checking stream and events in schema

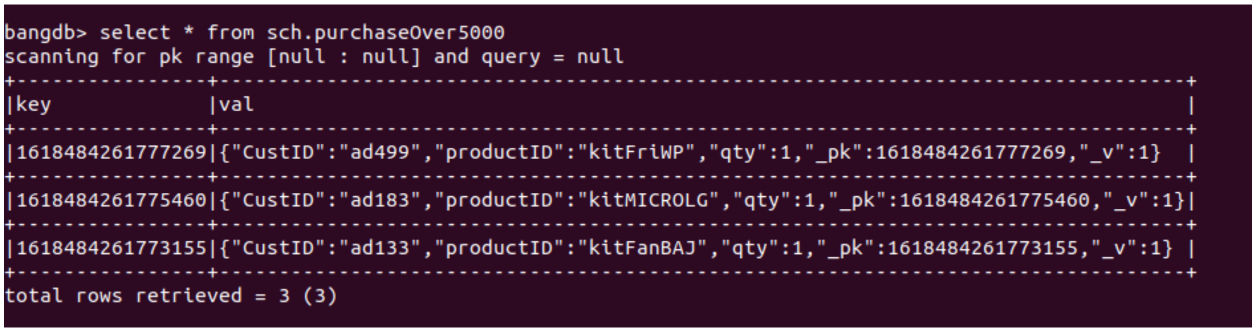
To check the event count in a schema for every stream present in it we use the command "**select stream from sch**". As we know we have pushed 20 events to the customer stream and 3 events should go to the filter stream based on our filter condition.



```
bangdb> select stream from sch
scanning for pk range [null : null] and query = null
scanning for pk range [null : null] and query = null
+----------------+----+----------+
|stream          |type|num_events|
+----------------+----+----------+
|customers       |1   |        20|
+----------------+----+----------+
|purchaseOver5000|2   |         3|
+----------------+----+----------+
successful in getting the streams [ 2 num ] info for schema [ sch ]
```

- To retrieve data from filter stream

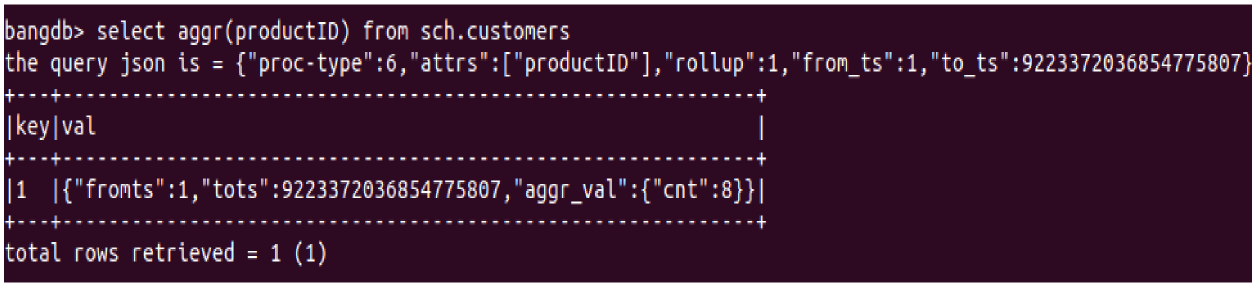
Let’s check if the data in stream is according to the condition or not. To do this the query is “**select * from sch.purchaseOver5000**”



```
bangdb> select * from sch.purchaseOver5000
scanning for pk range [null : null] and query = null
+----------------+---------------------------------------------------------------------------------+
|key             |val                                                                              |
+----------------+---------------------------------------------------------------------------------+
|1618484261777269|{"CustID":"ad499","productID":"kitFriWP","qty":1,"_pk":1618484261777269,"_v":1}  |
+----------------+---------------------------------------------------------------------------------+
|1618484261775460|{"CustID":"ad183","productID":"kitMICROLG","qty":1,"_pk":1618484261775460,"_v":1}|
+----------------+---------------------------------------------------------------------------------+
|1618484261773155|{"CustID":"ad133","productID":"kitFanBAJ","qty":1,"_pk":1618484261773155,"_v":1} |
+----------------+---------------------------------------------------------------------------------+
total rows retrieved = 3 (3)
```

- Checking total number of customers

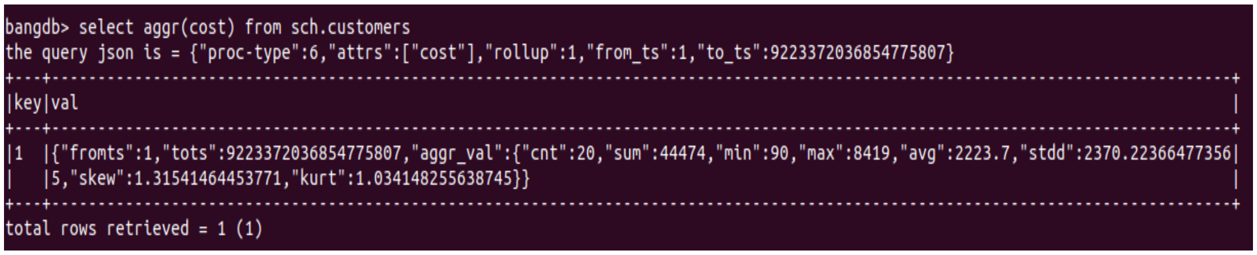
As we have selected stat 1 (which is the count value) for CustID. We will be checking the stat value for CustID. The query to do this is “**select aggr(CustID) from sch.customers**”.



```
bangdb> select aggr(productID) from sch.customers
the query json is = {"proc-type":6,"attrs":["productID"],"rollup":1,"from_ts":1,"to_ts":9223372036854775807}
+---+---------------------------------------------------------------+
|key|val                                                            |
+---+---------------------------------------------------------------+
|1  |{"fromts":1,"tots":9223372036854775807,"aggr_val":{"cnt":8}}|
+---+---------------------------------------------------------------+
total rows retrieved = 1 (1)
```

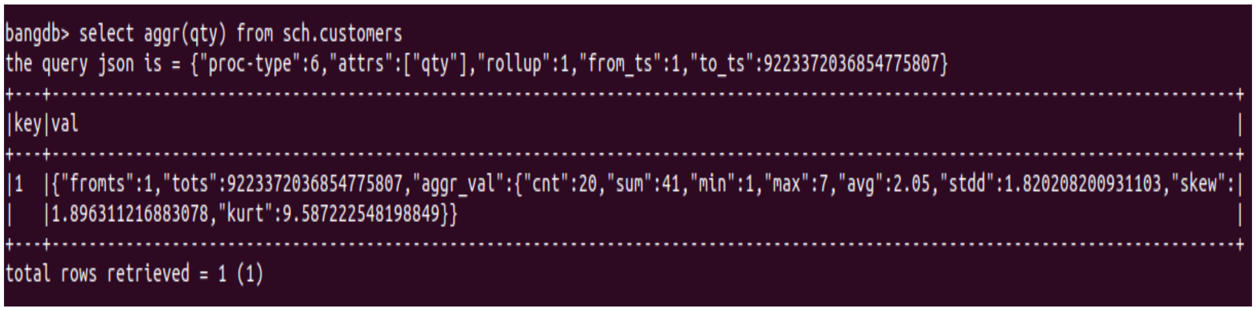
- Monitoring qty attribute and Cost attribute

Here, monitoring means we are observing the distribution of attribute qty and cost. This is achieved by enabling running stat on the them which we have already done when we were creating the schema. Command to check is “**select aggr(cost) from sch.customers**” and “**select aggr(qty) from sch.customers**”



```
bangdb> select aggr(cost) from sch.customers
the query json is = {"proc-type":6,"attrs":["cost"],"rollup":1,"from_ts":1,"to_ts":9223372036854775807}
+---+----------------------------------------------------------------------------------------------------------------------+
|key|val                                                                                                                   |
+---+----------------------------------------------------------------------------------------------------------------------+
|1  |{"fromts":1,"tots":9223372036854775807,"aggr_val":{"cnt":20,"sum":44474,"min":90,"max":8419,"avg":2223.7,"stdd":2370.22366477356|
|   |5,"skew":1.31541464453771,"kurt":1.034148255638745}}                                                                    |
+---+----------------------------------------------------------------------------------------------------------------------+
total rows retrieved = 1 (1)
```

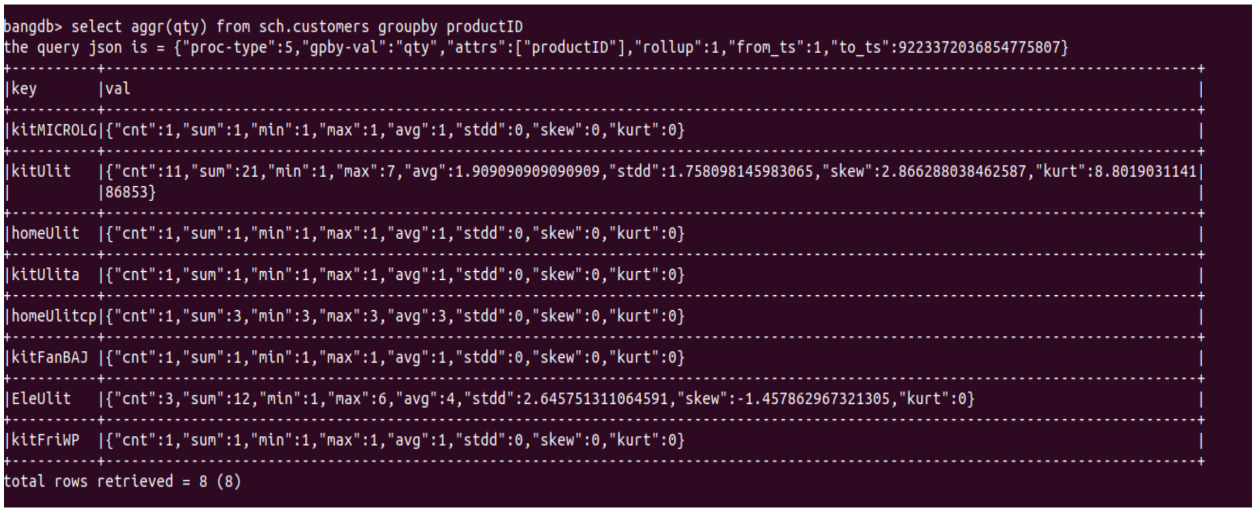
And



```
bangdb> select aggr(qty) from sch.customers
the query json is = {"proc-type":6,"attrs":["qty"],"rollup":1,"from_ts":1,"to_ts":9223372036854775807}
+---+---------------------------------------------------------------------------------------------------------+
|key|val                                                                                                      |
+---+---------------------------------------------------------------------------------------------------------+
|1  |{"fromts":1,"tots":9223372036854775807,"aggr_val":{"cnt":20,"sum":41,"min":1,"max":7,"avg":2.05,"stdd":1.820208200931103,"skew":|
|   |1.896311216883078,"kurt":9.587222548198849}}                                                              |
+---+---------------------------------------------------------------------------------------------------------+
total rows retrieved = 1 (1)
```

- Checking Group by

To check the group-by stats the query is “**select aggr(qty) from sch.customers groupby productID”**



```
bangdb> select aggr(qty) from sch.customers groupby productID
the query json is = {"proc-type":5,"gpby-val":"qty","attrs":["productID"],"rollup":1,"from_ts":1,"to_ts":9223372036854775807}
+----------+------------------------------------------------------------------------------------------------------------------+
|key       |val                                                                                                               |
+----------+------------------------------------------------------------------------------------------------------------------+
|kitMICROLG|{"cnt":1,"sum":1,"min":1,"max":1,"avg":1,"stdd":0,"skew":0,"kurt":0}                                             |
+----------+------------------------------------------------------------------------------------------------------------------+
|kitUlit   |{"cnt":11,"sum":21,"min":1,"max":7,"avg":1.909090909090909,"stdd":1.758098145983065,"skew":2.866288038462587,"kurt":8.8019031141|
|          |86853}                                                                                                            |
+----------+------------------------------------------------------------------------------------------------------------------+
|homeUlit  |{"cnt":1,"sum":1,"min":1,"max":1,"avg":1,"stdd":0,"skew":0,"kurt":0}                                             |
+----------+------------------------------------------------------------------------------------------------------------------+
|kitUlita  |{"cnt":1,"sum":1,"min":1,"max":1,"avg":1,"stdd":0,"skew":0,"kurt":0}                                             |
+----------+------------------------------------------------------------------------------------------------------------------+
|homeUlitcp|{"cnt":1,"sum":3,"min":3,"max":3,"avg":3,"stdd":0,"skew":0,"kurt":0}                                             |
+----------+------------------------------------------------------------------------------------------------------------------+
|kitFanBAJ |{"cnt":1,"sum":1,"min":1,"max":1,"avg":1,"stdd":0,"skew":0,"kurt":0}                                             |
+----------+------------------------------------------------------------------------------------------------------------------+
|EleUlit   |{"cnt":3,"sum":12,"min":1,"max":6,"avg":4,"stdd":2.645751311064591,"skew":-1.457862967321305,"kurt":0}            |
+----------+------------------------------------------------------------------------------------------------------------------+
|kitFriWP  |{"cnt":1,"sum":1,"min":1,"max":1,"avg":1,"stdd":0,"skew":0,"kurt":0}                                             |
+----------+------------------------------------------------------------------------------------------------------------------+
total rows retrieved = 8 (8)
```

Now let us consider a situation where we want to know what was the purchase cost a customer paid the first time when he or she visited

- Adding a ref

Enter the command **“create schema sch”** and follow the workflow

1. **create schema sch**
2. **What's the name of the stream that you wish to add?: customers**
   [name of the stream to which this operation will be attached]
3. **What would you like to add (press Enter when done) [attr (1) | catr (2) | refr (3) | gpby (4) | fltr (5) | join (6) | entity (7) | cep (8) | notifs (9)]: 3**
   [the operation to be attached]

   add refers (refr)...

4. **refr name: ref1 [Name for this ref operation]**

5. **enter name of input attribute that will get created after referring the other stream**: firstcost
   [new attribute]

6. **enter refr (other stream) stream name:** customers
   [referred stream]

7. **enter refr attribute name, the attribute if refer condition is satisfied:** cost
   [referred attribute]

8. **enter name of condition attribute need to join two events:** CustID
   [attribute used in condition]

9. **enter opid (operation name)[ATTR (a) | FIXED (f) | MATH_EXP (m) | HYBRID (h)]:** a
   [comparing with]

10. **enter name of arguments for joining (press Enter once done):** CustID
    [the other condition attribute]

11. **enter the comparison operators [EQ | NE | GT | LT] for join**: EQ
    [comparing operator]

add another refr ? [yes | no]:no

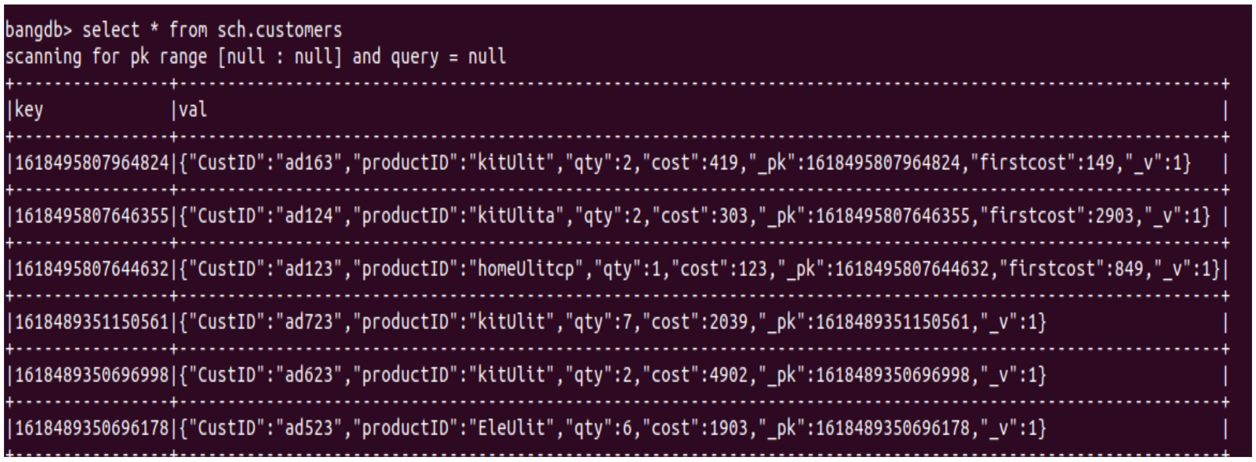
Save the schema

Inserting data again

```
insert into sch.customers values null {"CustID":"ad123","productID":"homeUlitcp","qty":1,"cost":123}
insert into sch.customers values null {"CustID":"ad124","productID":"kitUlita","qty":2,"cost":303}
insert into sch.customers values null {"CustID":"ad163","productID":"kitUlit","qty":2,"cost":419}
```

- Retrieving data

Select data from customer stream to check ref attribute. Enter command “**select * from sch.customers**”



```
bangdb> select * from sch.customers
scanning for pk range [null : null] and query = null
+----------------+------------------------------------------------------------------------------------------------------------+
|key             |val                                                                                                         |
+----------------+------------------------------------------------------------------------------------------------------------+
|1618495807964824|{"CustID":"ad163","productID":"kitUlit","qty":2,"cost":419,"_pk":1618495807964824,"firstcost":149,"_v":1}   |
+----------------+------------------------------------------------------------------------------------------------------------+
|1618495807646355|{"CustID":"ad124","productID":"kitUlita","qty":2,"cost":303,"_pk":1618495807646355,"firstcost":2903,"_v":1} |
+----------------+------------------------------------------------------------------------------------------------------------+
|1618495807644632|{"CustID":"ad123","productID":"homeUlitcp","qty":1,"cost":123,"_pk":1618495807644632,"firstcost":849,"_v":1}|
+----------------+------------------------------------------------------------------------------------------------------------+
|1618489351150561|{"CustID":"ad723","productID":"kitUlit","qty":7,"cost":2039,"_pk":1618489351150561,"_v":1}                  |
+----------------+------------------------------------------------------------------------------------------------------------+
|1618489350696998|{"CustID":"ad623","productID":"kitUlit","qty":2,"cost":4902,"_pk":1618489350696998,"_v":1}                  |
+----------------+------------------------------------------------------------------------------------------------------------+
|1618489350696178|{"CustID":"ad523","productID":"EleUlit","qty":6,"cost":1903,"_pk":1618489350696178,"_v":1}                  |
+----------------+------------------------------------------------------------------------------------------------------------+
```

Let's say that we want a notification when the first cost and cost are greater than 5000.

- Adding a CEP with notification

We will now attach a cep (complex event processing) query which will send notification when the first cost and cost for a customer is greater than 5000.

1. **Create schema sch**

2. **What's the name of the stream that you wish to add?: customers**
   [name of the stream to which this operation will be attached]

3. **What would you like to add (press Enter when done) [attr (1) | catr (2) | refr (3) | gpby (4) | fltr (5) | join (6) | entity (7) | cep (8) | notifs (9)]:** 8
   [the operation to be attached]

add complex event query (cepq)...

4. **cep query name: cep1** - [name for this cep operation]

5. **cep join type [join once (1) | passive join (2) | passive join with latest (3) | active join (4) | active join with latest (5) | simple join (6)]: 6**
   [join type, here we are performing simple join]

6. **enter the temporal locality time window (in seconds): 846000**
   [the duration for this operation]

Below section is same as for filter

7. **add fqry? [yes | no]: yes**
8. **enter type of fqry [query (1) | udf (2)]: 1**
9. **enter the filter key type [ATTR (1) | computed (3)] (or Enter for default (1))]: 1**
10. **enter the name of the attribute for comparison from input stream: cost**

11. **enter value type [attr val,(ex; $a) (1) | computed (3) | string (5) | long (9) | double (11)]: 11**
12. **enter value: 5000**
13. **enter comparison operation [GT | GTE | LT | LTE | EQ | NE]: GT**

add another query ? [yes | no]: yes--adding one more condition

14. **enter the filter key type [ATTR (1) | computed (3)]: 1**
15. **enter the name of the attribute for comparison from input stream: firstcost**
16. **enter value type [attr val,(ex; $a) (1) | computed (3) | string (5) | long (9) | double (11)]: 11**
17. **enter value: 5000**
18. **enter comparison operation [GT | GTE | LT | LTE | EQ | NE]: GT**

Add notification for this event

19. **Do you want to send notification as well? [yes | no]: yes**
20. **create a new notification template? [yes | no]: yes**

create new notification template:

21. **notification name: discount**
    [name for the notification]

22. **notification id: 500**
    [giving an ID to the notification]

23. **enter notification msg: eligible_for_discount**
    [Message for this notification]

24. **frequency in seconds: 5**
    [minimum duration between two notifications]

25. **priority of the notification [1, 2 or 3] (1 is highest): 1**
    [ranking the importance for the notification]

26. **enter mail ids [whom notificaions will be sent to]: admin@xxx.com**
    [Send by user]

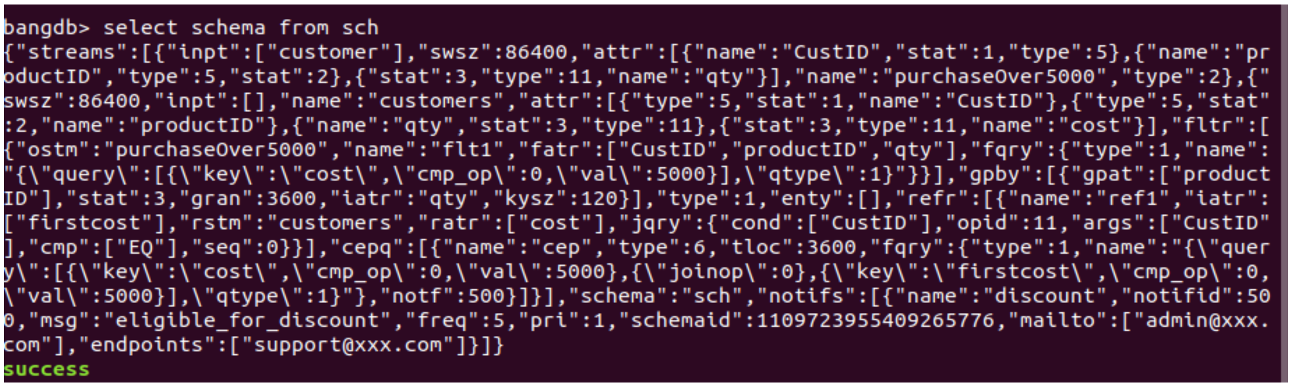
27. **enter API end points [whom notifications will be sent to]:support@xxx.com**
    [receiver user]

add another notification ? [yes | no]: no
enter the notification id: 500----[Attaching the notification to this event]

Press enter for all other inputs and Save the schema

- To check the schema

We will check if the schema got registered or not by checking the schema present. Query **“select schema from sch**”



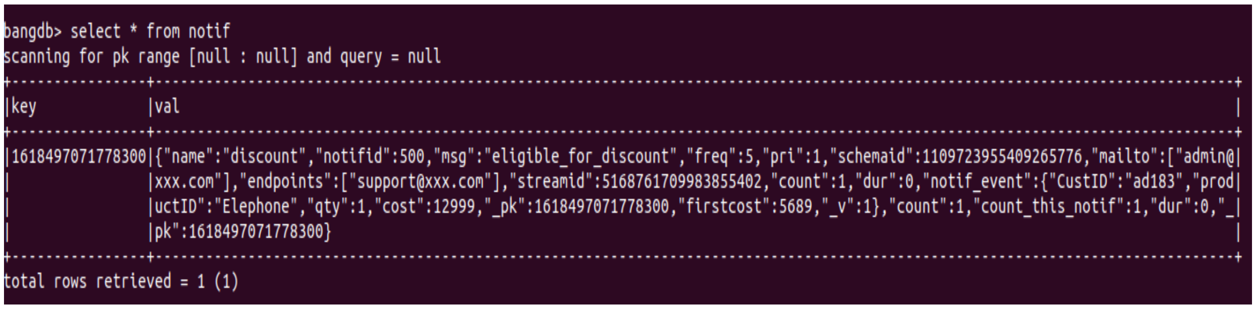
```
bangdb> select schema from sch
{"streams":[{"inpt":["customer"],"swsz":86400,"attr":[{"name":"CustID","stat":1,"type":5},{"name":"pr
oductID","type":5,"stat":2},{"stat":3,"type":11,"name":"qty"}],"name":"purchaseOver5000","type":2},{"
swsz":86400,"inpt":[],"name":"customers","attr":[{"type":5,"stat":1,"name":"CustID"},{"type":5,"stat"
:2,"name":"productID"},{"name":"qty","stat":3,"type":11},{"stat":3,"type":11,"name":"cost"}],"fltr":[
{"ostm":"purchaseOver5000","name":"flt1","fatr":["CustID","productID","qty"],"fqry":{"type":1,"name":
"{\"query\":[{\"key\":\"cost\",\"cmp_op\":0,\"val\":5000}],\"qtype\":1}"}}],"gpby":[{"gpat":["product
ID"],"stat":3,"gran":3600,"iatr":"qty","kysz":120}],"type":1,"enty":[],"refr":[{"name":"ref1","iatr":
["firstcost"],"rstm":"customers","ratr":["cost"],"jqry":{"cond":["CustID"],"opid":11,"args":["CustID"
],"cmp":["EQ"],"seq":0}}],"cepq":[{"name":"cep","type":6,"tloc":3600,"fqry":{"type":1,"name":"{\"quer
y\":[{\"key\":\"cost\",\"cmp_op\":0,\"val\":5000},{\"joinop\":0},{\"key\":\"firstcost\",\"cmp_op\":0,
\"val\":5000}],\"qtype\":1}"},"notf":500}]}],"schema":"sch","notifs":[{"name":"discount","notifid":50
0,"msg":"eligible_for_discount","freq":5,"pri":1,"schemaid":1109723955409265776,"mailto":["admin@xxx.
com"],"endpoints":["support@xxx.com"]}]}
success
```

Now we will insert data

```
insert into sch.customers values null {"CustID":"ad183","productID":"Elephone","qty":1,"cost":12999}
insert into sch.customers values null {"CustID":"ad193","productID":"homeUlit","qty":1,"cost":542}
insert into sch.customers values null {"CustID":"ad223","productID":"homeUlit","qty":1,"cost":84}
```

- List of notification generated

On inserting the above data, we can say that only one customer satisfies the condition for notification so only one notification should be generated. To check enter command “**select * from notif**”



```
bangdb> select * from notif
scanning for pk range [null : null] and query = null
+----------------+------------------------------------------------------------------------------------------------------------+
|key             |val                                                                                                         |
+----------------+------------------------------------------------------------------------------------------------------------+
|1618497071778300|{"name":"discount","notifid":500,"msg":"eligible_for_discount","freq":5,"pri":1,"schemaid":1109723955409265776,"mailto":["admin@|
|                |xxx.com"],"endpoints":["support@xxx.com"],"streamid":5168761709983855402,"count":1,"dur":0,"notif_event":{"CustID":"ad183","prod|
|                |uctID":"Elephone","qty":1,"cost":12999,"_pk":1618497071778300,"firstcost":5689,"_v":1},"count":1,"count_this_notif":1,"dur":0,"_|
|                |pk":1618497071778300}                                                                                                       |
+----------------+------------------------------------------------------------------------------------------------------------+
total rows retrieved = 1 (1)
```

Let’s consider a different scenario. Suppose we have a schema which has data about longitude and latitude and we have to find all the events near a given location.

Let the schema name be schgeo and the stream name be locations.

- Registering a already existing schema

We will register the schema given below directly from CLI. Users can also below schema if they wish to try.

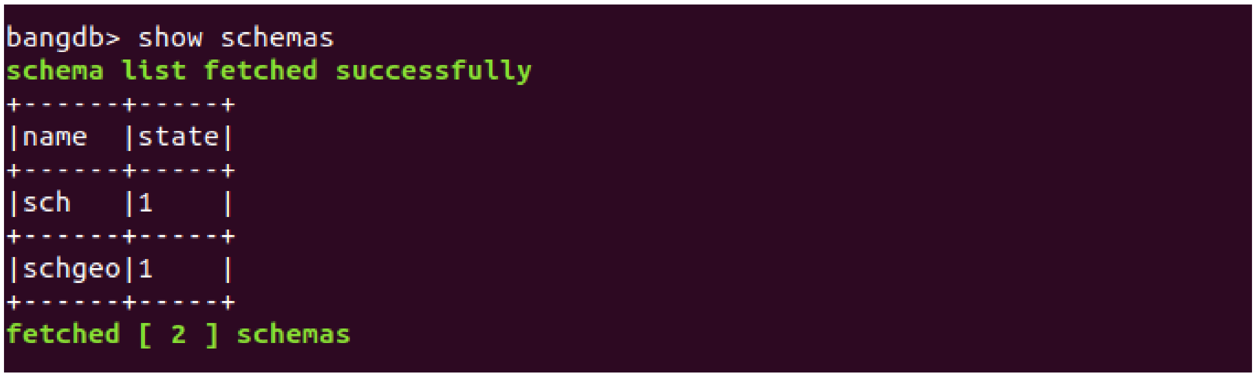
```
{
  "streams" : [
          {
          "type" : 1,"inpt" : [],"name" : "locations","swsz" : 3600,
          "attr" : [{"name" : "lat","type" : 11},{"name" : "lon","type" : 11}]
          }
  ],
  "schema" : "schgeo"
}
```

Copy the above schema to a file. We have saved the above schema to a filename schGEO.json.Note we have to provide the file path.

To register an already existing schema the command is "**register schema schGEO.json**"

- Listing all schemas present

After we have register the schema we will check to see all the schema's present and to do this the query is "**show schemas**"



```
bangdb> show schemas
schema list fetched successfully
+------+-----+
|name  |state|
+------+-----+
|sch   |1    |
+------+-----+
|schgeo|1    |
+------+-----+
fetched [ 2 ] schemas
```

- Adding Catr to get geo indexing

To find event within some distance from given location we need geo index which we will calculated using in build option in catr.

The steps are the same as adding any other operation to stream. Enter "**create schema schgeo**" and workflow will start. Users can follow the steps given below.

1. **create schema schgeo**
2. **what's the name of the stream that you wish to add?: locations**
   [Enter the stream to which we want to attach the operation]
3. **What would you like to add (press Enter when done) [attr (1) | catr (2) | refr (3) | gpby (4) | fltr (5) | join (6) | entity (7) | cep (8) | notifs (9)]: 2**
   [Select the operation to be attached, here it 2 - catr]

add computed attributes (catr)

4. **attribute name (press Enter to end): geoindex**
   [name of the attribute which will hold geo index values]

5. **attribute type (press Enter for default (5)) [string(5)|long(9)|double (11)]: 5**
   [data type for the new catr attribute]

6. **enter the name of the intended operation from the above default ops: GEOHASH**
   [for Geo index the in build operation is GEOHASH, a list of all operation will be displayed for users to select from]

7. **enter the input attributes on which this ops will be performed: lat**
8. **enter the input attributes on which this ops will be performed: lon**
   [attributes required for this operation]

Press enter for all other options and Save the schema

- Inserting some values.

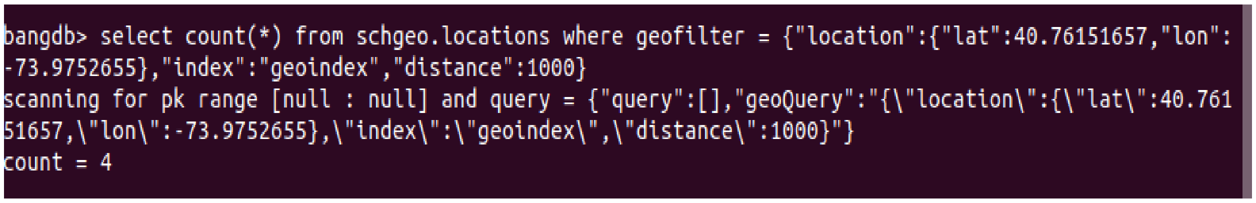
Users can also insert the same values.

```
insert into schgeo.locations values null {"lon":-73.98174286,"lat":40.71915817}
insert into schgeo.locations values null {"lon":-73.98275453,"lat":40.71915818}
insert into schgeo.locations values null {"lon":-73.98174286,"lat":40.71915817}
insert into schgeo.locations values null {"lon":-73.98508453,"lat":40.74716568}
insert into schgeo.locations values null {"lon":-73.97333527,"lat":40.76407242}
insert into schgeo.locations values null {"lon":-73.99310303,"lat":40.75263214}
insert into schgeo.locations values null {"lon":-73.98229218,"lat":40.75133133}
insert into schgeo.locations values null {"lon":-73.96527863,"lat":40.80104065}
insert into schgeo.locations values null {"lon":-73.97010803,"lat":40.75979996}
insert into schgeo.locations values null {"lon":-73.99373627,"lat":40.74176025}
insert into schgeo.locations values null {"lon":-73.98544312,"lat":40.73571014}
insert into schgeo.locations values null {"lon":-73.97686005,"lat":40.68337631}
insert into schgeo.locations values null {"lon":-73.9697876,"lat":40.75758362}
insert into schgeo.locations values null {"lon":-73.99397278,"lat":40.74086761}
insert into schgeo.locations values null {"lon":-74.00531769,"lat":40.72866058}
insert into schgeo.locations values null {"lon":-73.99013519,"lat":40.74885178}
insert into schgeo.locations values null {"lon":-73.9595108,"lat":40.76280975}
insert into schgeo.locations values null {"lon":-73.99025726,"lat":40.73703384}
insert into schgeo.locations values null {"lon":-73.99495697,"lat":40.745121}
insert into schgeo.locations values null {"lon":-73.93579865,"lat":40.70730972}
```

- Count the number of events based within 1000 meters from given locations

The query to do this “**select count(*) from schgeo.locations where geofilter = {"location":{"lat":40.76151657,"lon":-73.9752655},"index":"geoindex","distance":1000}**”

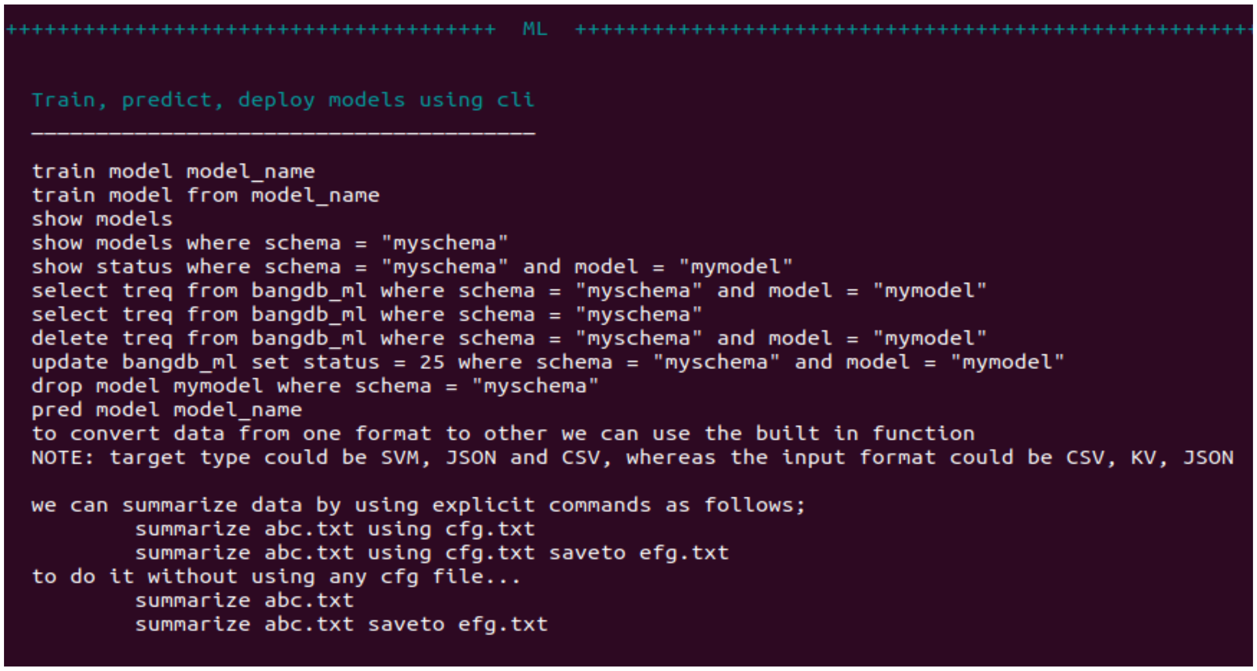
Here, the values of lat and lon are the given locations.



```
bangdb> select count(*) from schgeo.locations where geofilter = {"location":{"lat":40.76151657,"lon":
-73.9752655},"index":"geoindex","distance":1000}
scanning for pk range [null : null] and query = {"query":[],"geoQuery":"{\"location\":{\"lat\":40.761
51657,\"lon\":-73.9752655},\"index\":\"geoindex\",\"distance\":1000}"}
count = 4
```

Machine Learning on BangDB

Users can check ML related commands if they enter "**help ml**" as shown below. We will be explaining ML commands with an example so that users can also try it using the commands and data give for this Example.



```
+++++++++++++++++++++++++++++++++++++++++   ML   ++++++++++++++++++++++++++++++++++++++++++++++++++++++++


  Train, predict, deploy models using cli
  ----------------------------------------

  train model model_name
  train model from model_name
  show models
  show models where schema = "myschema"
  show status where schema = "myschema" and model = "mymodel"
  select treq from bangdb_ml where schema = "myschema" and model = "mymodel"
  select treq from bangdb_ml where schema = "myschema"
  delete treq from bangdb_ml where schema = "myschema" and model = "mymodel"
  update bangdb_ml set status = 25 where schema = "myschema" and model = "mymodel"
  drop model mymodel where schema = "myschema"
  pred model model_name
  to convert data from one format to other we can use the built in function
  NOTE: target type could be SVM, JSON and CSV, whereas the input format could be CSV, KV, JSON

  we can summarize data by using explicit commands as follows;
          summarize abc.txt using cfg.txt
          summarize abc.txt using cfg.txt saveto efg.txt
  to do it without using any cfg file...
          summarize abc.txt
          summarize abc.txt saveto efg.txt
```

Let's solve the problem stated below:

Predict whether a person will be a loan defaulter. It's a binary classification problem.

Algorithm: We will be using SVC for the above problem

The training and testing data is present at the end of the document.

Let's first register the schema. Users can use the schema present below (copy the schema to a file then register the schema from CLI)

We have saved the below schema to a file == sml.json

Command to register schema ”**register schema sml.json”.** Here sml.json is the name of the file which contains the schema.

```
{
  "schema": "sml",
  "streams": [
            {
            "name": "application",
            "type": 1,
            "attr": [{"name" : "loan_status","type" : 9},{"type" : 11,"name" : "load_amt"},{"name" :
"funded_amnt","type" : 11},{"name" : "funded_amnt_inv","type" : 11},{"name" : "term","type" :
9},{"name" : "int_rate","type" : 11},{"name" : "installment","type" : 11},{"type" : 9,"name" :
"grade"},{"name" : "home_ownership","type" : 9},{"name" : "verification","type" : 9},{"name" :
"pymnt_plan","type" : 9},{"name" : "mths_lastdelinq","type" : 11},{"name" : "revol_bal","type" :
11},{"type" : 11,"name" : "out_prncp"},{"name" : "out_prncp_inv","type" : 11},{"name" :
"total_pymnt","type" : 11},{"type" : 11,"name" : "total_pymnt_inv"},{"type" : 11,"name" :
"total_rec_prncp"},{"name" : "total_rec_int","type" : 11},{"type" : 11,"name" :
"total_rec_late_fee"},{"name" : "recoveries","type" : 11},{"type" : 11,"name" :
"collection_recovery_fee"},{"type" : 11,"name" : "last_pymnt_amnt"},{"type" : 11,"name" :
"policy_code"},{"name" : "tot_coll_amt","type" : 11},{"name" : "tot_cur_bal","type" : 11},{"type" :
11,"name" : "total_rev_hi_lim"}
            ],
            "swsz": 86400,
            "inpt": []
            }
  ]
}
```

1. Summarizing the training data

Before training a model we should study the data to check for statistically significant of the attributes present. We will use the summarize command to get the statistics and correlations between the target attribute and other attributes.

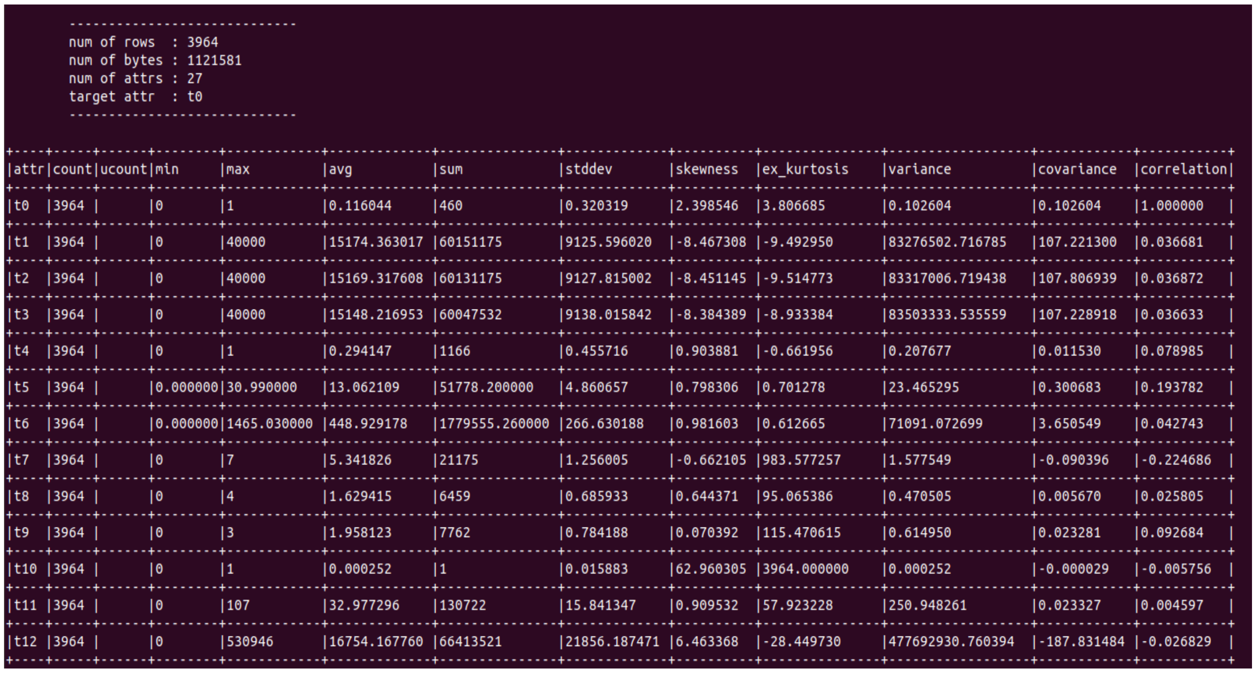
The query to do this “**summarize /home/iql0005/Desktop/load_train.csv**”--- this will start a workflow

Step1. summarize /home/iql0005/Desktop/load_train.csv
[enter the file location on local system]

Step2. Do you want to set the target attr index (or 'enter' if you wish to select the last one as target attr)? [yes | no]: yes
[We have to set the target attribute]

Step3. enter the target index [starts with 0 to n-1 cols]: 0
[As our target attribute is at position 0 in training file]

Output of the above query will be like



```
        ----------------------------
        num of rows  : 3964
        num of bytes : 1121581
        num of attrs : 27
        target attr  : t0
        ----------------------------

+----+-----+------+--------+-----------+------------+---------------+------------+----------+----------------+------------------+-----------+-----------+
|attr|count|ucount|min     |max        |avg         |sum            |stddev      |skewness  |ex_kurtosis     |variance          |covariance |correlation|
+----+-----+------+--------+-----------+------------+---------------+------------+----------+----------------+------------------+-----------+-----------+
|t0  |3964 |      |0       |1          |0.116044    |460            |0.320319    |2.398546  |3.806685        |0.102604          |0.102604   |1.000000   |
+----+-----+------+--------+-----------+------------+---------------+------------+----------+----------------+------------------+-----------+-----------+
|t1  |3964 |      |0       |40000      |15174.363017|60151175       |9125.596020 |-8.467308 |-9.492950       |83276502.716785   |107.221300 |0.036681   |
+----+-----+------+--------+-----------+------------+---------------+------------+----------+----------------+------------------+-----------+-----------+
|t2  |3964 |      |0       |40000      |15169.317608|60131175       |9127.815002 |-8.451145 |-9.514773       |83317006.719438   |107.806939 |0.036872   |
+----+-----+------+--------+-----------+------------+---------------+------------+----------+----------------+------------------+-----------+-----------+
|t3  |3964 |      |0       |40000      |15148.216953|60047532       |9138.015842 |-8.384389 |-8.933384       |83503333.535559   |107.228918 |0.036633   |
+----+-----+------+--------+-----------+------------+---------------+------------+----------+----------------+------------------+-----------+-----------+
|t4  |3964 |      |0       |1          |0.294147    |1166           |0.455716    |0.903881  |-0.661956       |0.207677          |0.011530   |0.078985   |
+----+-----+------+--------+-----------+------------+---------------+------------+----------+----------------+------------------+-----------+-----------+
|t5  |3964 |      |0.000000|30.990000  |13.062109   |51778.200000   |4.860657    |0.798306  |0.701278        |23.465295         |0.300683   |0.193782   |
+----+-----+------+--------+-----------+------------+---------------+------------+----------+----------------+------------------+-----------+-----------+
|t6  |3964 |      |0.000000|1465.030000|448.929178  |1779555.260000 |266.630188  |0.981603  |0.612665        |71091.072699      |3.650549   |0.042743   |
+----+-----+------+--------+-----------+------------+---------------+------------+----------+----------------+------------------+-----------+-----------+
|t7  |3964 |      |0       |7          |5.341826    |21175          |1.256005    |-0.662105 |983.577257      |1.577549          |-0.090396  |-0.224686  |
+----+-----+------+--------+-----------+------------+---------------+------------+----------+----------------+------------------+-----------+-----------+
|t8  |3964 |      |0       |4          |1.629415    |6459           |0.685933    |0.644371  |95.065386       |0.470505          |0.005670   |0.025805   |
+----+-----+------+--------+-----------+------------+---------------+------------+----------+----------------+------------------+-----------+-----------+
|t9  |3964 |      |0       |3          |1.958123    |7762           |0.784188    |0.070392  |115.470615      |0.614950          |0.023281   |0.092684   |
+----+-----+------+--------+-----------+------------+---------------+------------+----------+----------------+------------------+-----------+-----------+
|t10 |3964 |      |0       |1          |0.000252    |1              |0.015883    |62.960305 |3964.000000     |0.000252          |-0.000029  |-0.005756  |
+----+-----+------+--------+-----------+------------+---------------+------------+----------+----------------+------------------+-----------+-----------+
|t11 |3964 |      |0       |107        |32.977296   |130722         |15.841347   |0.909532  |57.923228       |250.948261        |0.023327   |0.004597   |
+----+-----+------+--------+-----------+------------+---------------+------------+----------+----------------+------------------+-----------+-----------+
|t12 |3964 |      |0       |530946     |16754.167760|66413521       |21856.187471|6.463368  |-28.449730      |477692930.760394  |-187.831484|-0.026829  |
+----+-----+------+--------+-----------+------------+---------------+------------+----------+----------------+------------------+-----------+-----------+
```

2. Training a model on BangDB

There are two ways to train a model on bangdb. One is to directly register the meta_data for training (we call it json request which contains all the details about the model) and second is to create mage_data for training by following the workflow on CLI..

Here we will be training the model using both methods.

- ### Method : training model by following workflow on cli

On bangdb, we have the option of training a model from a file(file format can be libSVM, CSV or JSON) containing training data or from streaming data. Here we are training the model from a CSV file.

1. **Enter command "train model model_name" :- train model model_dft**
   Here the workflow starts. User just have to enter the training details:

STEPS AND PARAMETER EXPLANATIONS FOR THE CLI WORKFLOW FOR TRAINING

2. **What's the name of the schema for which you wish to train the model?: sml**
3. **do you wish to setup the ml env or use the default[yes | no]: no**
4. **do you wish to read the earlier saved ml schema for editing/adding? [yes | no]: no**

Here we will get a list of all the algorithms supported. As we have to perform Classification, we will be selecting classification option for next command

5. **what's the algo would you like to use (or Enter for default (1)): 1**
6. **svm type [C_SVC (0) | NU_SVC (1) | ONE_CLASS (2)] (press enter for default (0)): 0**

   [For classification, we have 3 types of SVC. According to the form of error function, SVC models can be classified into Two distinct groups: Classification SVM Type 1 (also known as C-SVM classification) and Classification SVM Type 2 (also known as nu-SVM classification). First and second are for both multi and single classification but the third one “ONE-CLASS” is for binary classifications only **]**

7. **kernel type [LINEAR (0) | POLY (1) | RBF (2) | SIGMOID (3)] (press enter for default (2)): 2”**
   [set type of kernel function
   0 -- linear: u'*v
   1 -- polynomial: (gamma*u'*v + coef0)^degree
   2 -- radial basis function: exp(-gamma*|u-v|^2)
   3 -- sigmoid: tanh(gamma*u'*v + coef0)]

8. **degree (press enter for default (3): 2**
   [polynomial kernel parameter]

9. **Enter gamma (or press enter for default (0.001)): 0.002”**
   [kernel parameter]

10. **Enter C (or press enter for default):** 2
   [algorithm parameter : Penalty factor of misclassification.]

11. **Enter weight (or press enter for default): 1:07**
   [algorithm parameter : set the parameter C of class i to weight*C, for C-SVC]

12. **Enable shrinking? [yes | no]: yes**
   [The shrinking is there to save the training time.**]**

13. **Enable probability? [yes | no]: no**
   [For probability estimate **]**

14. **What's the stopping criteria (eps) (or press enter for default (0.001)): 0.1 “**
   [Set tolerance of termination criterion **]**

15. **What's the input (training data) source?[local file (1)|file on BRS (2)|stream (3)]**: 1
   [Here, users have to specify the source of data whether it's a file store in the local system or in BRS or its streaming data. For this use-case, the training file is store in local system **]**

16. **Enter the file name for upload (along with full path):**
   /home/iql0005/Desktop/Classification/Binary_classification/loan_train.csv”

17. **Enter the data format for the training data [libSVM (1) | CSV (2) | JSON (3)] (enter for default): 2**
   [Here, users have to define the format of the training data file]

18. **What is the separator (SEP) for the csv file? (press Enter for default ',' (comma) else type it):** ,
   [As we are training from CSV file format with separator “ , ”]

19. **What is the target index (to select the target attribute/val): 0**
    [Here, we have no specify position of target attribute]

20. **What's the training speed you wish to select [Very fast (1) | fast (2) | medium (3) | slow (4) | very slow (5)] (or Enter for default (1)): 4**
    [Here, usres defines the speed for reaching optimization, higher the speed training time is longer]

BangDB deals with categorical data on its own by converting categorical to numerical, users just have to select the proper attribute type in the option below.

21. **What's the attribute type [NUM (1) | STRING (2) | HYBRID (3)]: 3**
    [Users have to specify the nature of attributes present in the training file. If all attributes are numerical then select option 1, if all are string select 2 and for both categorical and numerical select 3.]

22. **Do you wish to scale the data? [yes | no]: yes**
    [Scaling data is very necessary for SVM]

23. **Do you wish to tune the params? [yes | no]: yes**
    [Here, If user select yes for tuning, in the backend grid search is performed to find best values for gamma and C]

Next, we need to do the mapping. This means we need to provide attribute name and its position in the training file and define which is the target attribute (target attribute is represent by position 0)

24. **Enable attr name: loan_status**
    enable attr position: 0
    do you wish to add more attributes? [yes | no]: **yes**”

25. **Enable attr name: loan_amt**
    enable attr position: 1
    do you wish to add more attributes? [yes | no]: **yes**”
    …………..
    …………..
    Aftering entering all attributes

26. **Do you wish to add more attributes? [yes | no]: no**”

27. **Do you wish to add external udf to do some computations before the training? [yes | no]: no**

Here, we can view the meta_data which we created for training.

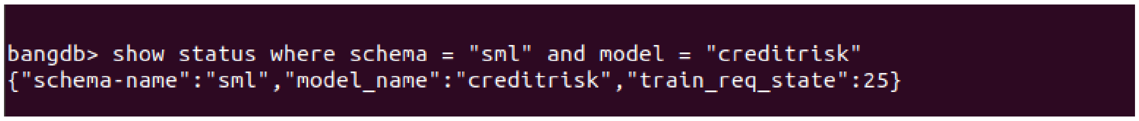
updated schema :

```
{
    "attr_list" : [
            {
               "name" : "loan_status",
               "position" : 0
            },
            {
                "name" : "loan_amt",
                 "position" : 1
            },
            …………..
            ],
   "model_name" : "creditRisk",
  "scale" : 1,
  "training_details" : {
            "expected_format" : "SVM",
            "file_size_mb" : 1,
            "SEP" : ",",
            "training_source" : "load_train.csv",
            "training_source_type" : 1,
            "input_format" : "CSV",
            "train_speed" : 5,
            "target_idx" : 0
  },
  "tune_params" : 1,
  "schema-name" : "sml",
  "attr_type" : 3,
     "algo_param" : {
           "gamma" : 0.002,
            "shrinking" : 0,
            "cost" : 2,
            "termination_criteria" : 0.001,
            "kernel_type" : 2,
            "svm_type" : 0,
            "probability" : 0,
            "degree" : 2
            }
```

28. Do you wish to start training now? [yes | no]: **yes**”

you may check the train status by using 'show train status' command ------training started

To check training status enter
Show status where schema = “schema_namel” and model = “model_name”
Training status 25 represents that the training is completed.



```
bangdb> show status where schema = "sml" and model = "creditrisk"
{"schema-name":"sml","model_name":"creditrisk","train_req_state":25}
```

- Method two : training model by uploading training request (training meta-data)

Step 1. Prepare a file containing training meta-data
Here, we have create a json file name model1.json and we have created the training request:
Training request: (users can copy and paste the below training request)

```
{"attr_type" : 1,"scale" : 1,
"attr_list" : [{"name" : "loan_status","position" : 0},{"position" : 1,"name" : "load_amt"},{"name" :
"funded_amnt","position" : 2},{"name" : "funded_amnt_inv","position" : 3},{"name" :
"term","position" : 4},{"name" : "int_rate","position" : 5},{"name" : "installment","position" :
6},{"position" : 7,"name" : "grade"},{"name" : "home_ownership","position" : 8},{"name" :
"verification","position" : 9},{"name" : "pymnt_plan","position" : 10},{"name" :
"mths_lastdelinq","position" : 11},{"name" : "revol_bal","position" : 12},{"position" : 13,"name" :
"out_prncp"},{"name" : "out_prncp_inv","position" : 14},{"name" : "total_pymnt","position" :
15},{"position" : 16,"name" : "total_pymnt_inv"},{"position" : 17,"name" :
"total_rec_prncp"},{"name" : "total_rec_int","position" : 18},{"position" : 19,"name" :
"total_rec_late_fee"},{"name" : "recoveries","position" : 20},{"position" : 21,"name" :
"collection_recovery_fee"},{"position" : 22,"name" : "last_pymnt_amnt"},{"position" : 23,"name" :
"policy_code"},{"name" : "tot_coll_amt","position" : 24},{"name" : "tot_cur_bal","position" :
25},{"position" : 26,"name" : "total_rev_hi_lim"}],
  "algo_param" : {"termination_criteria" : 0.001,"cost" : 100,"shrinking" : 0,"svm_type" :
0,"probability" : 0,"kernel_type" : 2},
  "algo_type" : "SVM",
  "model_name" : "creditrisk",
  "tune_params" : 1,
  "training_details" : {
          "SEP" : ",",
          "training_source_type" : 1,
          "file_size_mb" : 2,
          "target_idx" : 0,
          "training_source" : "load_train.csv",
          "train_speed" : 5,
          "input_format" : "CSV",
          "expected_format" : "SVM"},
  "schema-name" : "sml"}
```

Step 2. **Enter command " train model from model-meta-data**
Model-meta-data = is the location of the file containing the meta data for training with its file name.

From here the cli workflow will start

After entering the above command the training request in the file will be displayed on the screen
Step 3. **Cli will ask the path for training file on the system**

```
          upload file : loan_train.csv
             enter the path for upload (full path of the file):
             [ provide the training file path with file name]
```

Step 4. **Do you wish to start training now? [yes | no]: yes**

```
          [ Enter yes to start training]
```

schema [sml] registered successfully for training----------- on successful register of meta-data

The User has to understand that the training time taken depends on a lot of factors (like the parameter selected, size of data, bangdb setting etc..)

Once the training is started user can check the train status by using 'show train status' command

Step 5. show status where schema = "sml" and model = "creditrisk"

- Explaining Training Meta-Data

Json request Structure:

1. Schema name : Here its “sml”
2. Model name : creditrisk
3. algo_type: we are using SVM algorithm for model training
4. algo_param: Contains details about algorithm parameters--
   a. svm_type: 0 which is C-SVC
   b. "kernel_type":2, **[** set type of kernel function
      0 -- linear: u'*v
      1 -- polynomial: (gamma*u'*v + coef0)^degree
      2 -- radial basis function: exp(-gamma*|u-v|^2)
      3 -- sigmoid: tanh(gamma*u'*v + coef0) **]**
   c. "degree":2,   [polynomial kernel parameter]
   d. "gamma":0.01, [kernel parameter]
   e. "nu":0.2, [error tolerance]
   f. "shrinking":1,
   g. "probability":0, [Probability estimate]
   h. "termination_criteria":0.1 **[** Set tolerance of termination criterion **]**

5. Input_format : the format of training file and expected format: the format required for training.
   [Requirement for training “SVM” model is that the data set should be in libsvm file format. { Libsvm format== 23 1:23 2:55 3:655...} where target attribute should be place first therefore we have to convert csv or json file format to libsvm file format]

6. training_details
    a. training_source :loan_train.csv,
    b. file_size_mb :10,
    c. training_source_type :1, [source is file as we are training model using CSV file]
    d. train_speed :5, [Here, usres defines the speed for reaching optimization]
7. attr_type : 1 [num (1) when all attributes are numerical, when all attributes are string (2), hybrid(3) for mixed]
8. scale: 1, [Scale = 1 means we are performing scaling (converting attribute values between -1 to 1) on data, which is necessary for algorithms like SVM and KMeans but not mandatory.
9. tune_params : 1 [0 or 1, tune_params = 1, performing grid-search to find best values for C and gamma.]
10. attr_list : Contains list of input attribute for training and target attribute(position should be 0)

3. Deploying a Model for real time prediction

To Deploy the model, users just have to add Catr to the stream. [Note that the schema mentioned while training the model and schema of the stream where we are adding this model should be the same.]

Name of model is creditrisk and the schema is sml

Adding Catr which perform prediction using the above model

Step1. create schema sml --- this will start the workflow

Step2. What's the name of the stream that you wish to add?: application
[stream name which contains data for prediction]

Step3. What would you like to add (press Enter when done) [attr (1) | catr (2) | refr (3) | gpby (4) | fltr (5) | join (6) | entity (7) | cep (8) | notifs (9)]: 2
[For prediction we have to select catr]

add computed attributes (catr)...
Step4. attribute name (press Enter to end): predDefaulter
[name of the attribute which will stored predicted values]

Step5. attribute type (press Enter for default (5)) [string(5) | long(9) | double (11)]: 9
[data type]

Step6. enter the name of the intended operation from the above default ops: PRED
[Select operation, As we are performing prediction]

Step7. enter the name of the ML model: creditrisk
[Enter the model name]

Step8. enter the name of the algo [SVM|KMEANS|LIN|PY|IE|IE_SENT|IE_NER]: SVM
[Enter the algorithm used for training]

Step9. enter the attribute type for the model [NUM | STR | HYB]: NUM
[As all attributes are numerical]

Step10. what is the expected data format for the prediction[LIBSVM (0)|CSV (1)|JSON (3)]: 3
[As we are predicting on stream, we should select 3]

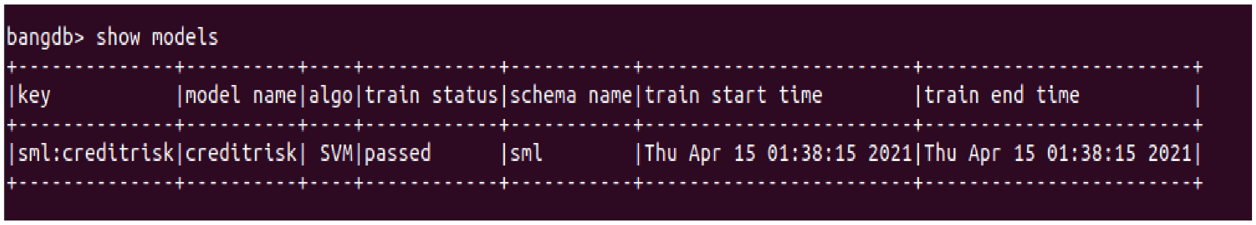
Step11. enter the input attributes on which this ops will be performed: loan_amt
....
[Enter all the attribute required for prediction]

Save the schema after change.Now as we push events into this stream, attribute predDefaulter will have the prediction for that event.

4. Query related ML models

- To get list of all model present

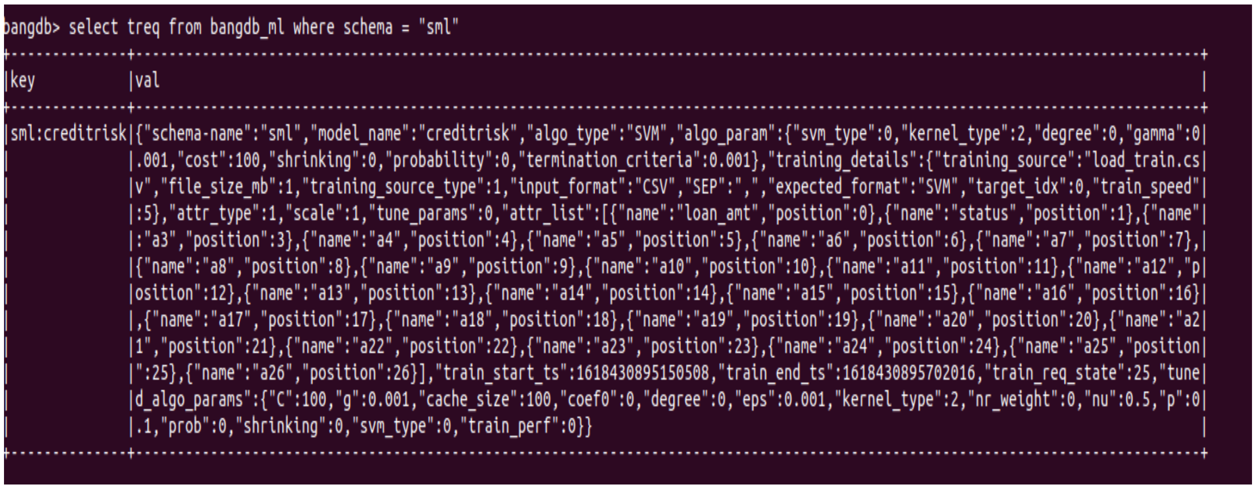
The query to do this is “**show models**”



```
bangdb> show models
+--------------+----------+----+------------+-----------+------------------------+------------------------+
|key           |model name|algo|train status|schema name|train start time        |train end time          |
+--------------+----------+----+------------+-----------+------------------------+------------------------+
|sml:creditrisk|creditrisk| SVM|passed      |sml        |Thu Apr  15 01:38:15 2021|Thu Apr  15 01:38:15 2021|
+--------------+----------+----+------------+-----------+------------------------+------------------------+
```

- To get training details from the above model

The query is “**select treq from bangdb_ml where schema = "sml"**



```
bangdb> select treq from bangdb_ml where schema = "sml"
+--------------+-------------------------------------------------------------------------------------------------------------------------------+
|key           |val                                                                                                                            |
+--------------+-------------------------------------------------------------------------------------------------------------------------------+
|sml:creditrisk|{"schema-name":"sml","model_name":"creditrisk","algo_type":"SVM","algo_param":{"svm_type":0,"kernel_type":2,"degree":0,"gamma":0|
|              |.001,"cost":100,"shrinking":0,"probability":0,"termination_criteria":0.001},"training_details":{"training_source":"load_train.cs|
|              |v","file_size_mb":1,"training_source_type":1,"input_format":"CSV","SEP":",","expected_format":"SVM","target_idx":0,"train_speed"|
|              |:5},"attr_type":1,"scale":1,"tune_params":0,"attr_list":[{"name":"loan_amt","position":0},{"name":"status","position":1},{"name"|
|              |:"a3","position":3},{"name":"a4","position":4},{"name":"a5","position":5},{"name":"a6","position":6},{"name":"a7","position":7},|
|              |{"name":"a8","position":8},{"name":"a9","position":9},{"name":"a10","position":10},{"name":"a11","position":11},{"name":"a12","p|
|              |osition":12},{"name":"a13","position":13},{"name":"a14","position":14},{"name":"a15","position":15},{"name":"a16","position":16}|
|              |,{"name":"a17","position":17},{"name":"a18","position":18},{"name":"a19","position":19},{"name":"a20","position":20},{"name":"a2|
|              |1","position":21},{"name":"a22","position":22},{"name":"a23","position":23},{"name":"a24","position":24},{"name":"a25","position|
|              |":25},{"name":"a26","position":26}],"train_start_ts":1618430895150508,"train_end_ts":1618430895702016,"train_req_state":25,"tune|
|              |d_algo_params":{"C":100,"g":0.001,"cache_size":100,"coef0":0,"degree":0,"eps":0.001,"kernel_type":2,"nr_weight":0,"nu":0.5,"p":0|
|              |.1,"prob":0,"shrinking":0,"svm_type":0,"train_perf":0}}                                                                         |
+--------------+-------------------------------------------------------------------------------------------------------------------------------+
```

- To test prediction on test data

Let’s predict for test data loan_test.csv using the above model.

Enter the command “ **pred model creditrisk**” -- this will start the cli workflow

```
  Step1. What's the name of the schema for which model was trained?: sml

  Step2. Do you wish to setup the ml env or use the default[ yes |  no ]: no

  Step3. Do you wish to see the train request? [ yes |  no ]: no

   model algo type is [ SVM ], it needs [ NUM ] data type with [ LIBSVM ] input data format

[Above comment will be displayed on the CLI, giving info about the model, its attribute type and because
the file format required for SVM is LIBSVM, therefore all SVM models show input data format as LIBSVM.
User can provide any one among the supported file formats, the file will be converted to required format]

  Step4. What is the input data format for the given pred file [ LIBSVM (0) | CSV (1) | JSON (3) ]: 1
         [As our test file is in CSV, we will select 1]

  Step5. What is the separator (SEP) for the csv file? (press Enter for default ',' (comma) else type it):
         [we have to mention the SEP for CSV test file]

  Step6. Do you wish to provide an attribute list? [ yes | no ]: no
         [in cases where arrangement of attributes are different while training and prediction]

  Step7. do you wish to consider the target (are you also supplying target value?) [ yes | no ]: no
         [we select this when have target within test data]

  Step 8. do you wish to pred the file? Or a single event? [ yes (file) |  no (single event) ]: yes”

  Step 9. Do you wish to upload the file? [ yes |  no ]: yes”

  Step 10. Enter the test file name for upload (along with full
path):/home/iql0005/Desktop/Classification/Binary_classification/loan_test.csv”

                              Following Message will be displayed on the screen
pred request =
{"input_format":"CSV","SEP":",","expected_format":"SVM","schema-name":"sml","model_name":"creditrisk
","algo_type":"SVM","attr_type":1,"consider_target":0,"data_type":1,"data":"load_test.csv"}
{"pred_file_out":"creditrisk__sml__load_test.csv.predict","errorcode":0}


         Once it's done, we can download the test file
```

- To download results for test file prediction

All data and training related files are stored in the “**ml_bucket_info**” bucket in BRS. To Download the query is

getfile predtestfile.csv from ml_bucket_info where filekey = "creditrisk__sml__load_test.csv.predict"