How to call other query in GSQL

Summary

In GSQL, inside one query body, we can call other queries. The called query must have a return value. The returned value can be used as an expression. This query-calling-query feature makes GSQL more expressive and modularized, thus improve user's productivity.

Note: we will use GSQL 101's social graph schema in our examples.

```
CREATE VERTEX person (PRIMARY_ID name STRING, name STRING, gender
STRING, age INT, state STRING)
CREATE UNDIRECTED EDGE friendship (FROM person, TO person, connect_day
DATETIME)
CREATE GRAPH social (person, friendship)
```

How to define a query with a return value

GSQL

```
#add returns(output_type)
CREATE QUERY SUBQUERY (input_parameter_list) FOR GRAPH social RETURNS
(output_type) { QUERY_BODY}
```

Example

GSQL

```
USE GRAPH social
CREATE QUERY one_hop_neighbor(VERTEX<person> p) FOR GRAPH social RETURNS
(SET<VERTEX<person>>) {

  Start = {p};

  Result = SELECT tgt
           FROM Start-(friendship:e)->person:tgt;

  RETURN Result;
}

#main query will call one_hop_neighbor
CREATE QUERY main(VERTEX<person> p) FOR GRAPH social{
  SetAccum<Vertex<person>> @@RetVal;
  Start = {p};

  @@RetVal += one_hop_neighbor(p);

  Result = {@@RetVal};

  PRINT Result;
}
#note if you install query one by one
#the called query should be installed first.
install query all
run query main("Tom")
```

The above query will have exact result as

GSQL

```
CREATE QUERY hello(VERTEX<person> p) FOR GRAPH social{
  Start = {p};
  Result = SELECT tgt
           FROM Start-(friendship:e)->person:tgt;
  PRINT Result;
}
```

- A called query can be a stand alone statement, in which the returned value is ignored by the main query.
- If there are PRINT statement in the called query, it won't be shown in the client side. Only PRINT in the main query will be visible.
- We have not exposed the recursive query capability to user now.

- Local accumulator computed in the main query is invisible in the called query.

- The supported input parameter types are

```
vertex, int, string, float, double, bool, bag and set.
```

- The supported return types are

```
 INT, STRING, FLOAT, DOUBLE, BOOL, SET, BAG
 #Below (1) the type cannot be a udt.  (2) GroupByAccum is not
supported.
 SumAccum<Type>, AvgAccum<Type>, MinAccum<Type>, MaxAccum<Type>,
SetAccum<Type>, MapAccum<Type>, BagAccum<Type>,   ListAccum<Type>,
ArrayAccum<Type>, HeapAccum<Type>, OrAccum, AndAccum,
```

More examples

GSQL

```
##############################################################################
##
# This query find a person p's 2-hop neighbors
# Purpose: show that we can pass vSet as a parameter to set<vertex<T>>
##############################################################################
##

use graph social
drop query main2
drop query two_hop_neighbor

CREATE QUERY two_hop_neighbor(Set<VERTEX<person>> nn, Vertex<person> p)
FOR GRAPH social RETURNS(Set<Vertex<person>>){

  Start = {nn};

  Result = SELECT tgt
           FROM Start-(friendship:e)->person:tgt
           WHERE tgt != p;

  RETURN Result;
}

#this query find p's two-hop neighbors without using
#local accumulator, since the called subquery won't see
#the local accumulator updated from the main query.
CREATE QUERY main2(VERTEX<person> p) FOR GRAPH social{
  int cnt = 0;
  SetAccum<VERTEX<person>> @@result;

  Start = {p};

  FirstNeighbors = SELECT tgt
                   FROM Start:s-(friendship:e)->person:tgt;

  # pass FirstNeighbors as parameter for a set<vertex<user>> type
parameter.
  @@result += two_hop_neighbor(FirstNeighbors, p);

  SecondNeighbors = {@@result};

  #secondNeighbors may include some first-hop neighbors, remove them
  Result = SecondNeighbors MINUS FirstNeighbors;

  PRINT Result;
}

INSTALL QUERY all
RUN QUERY main2("Tom")
```

GSQL

```
################################################################################
##
# This query find a person p's 2-hop neighbors
# Purpose: show that we can call subquery in ACCUM clause
################################################################################
##
use graph social
drop query main3
drop query one_step

CREATE QUERY one_step(VERTEX<person> p) FOR GRAPH social RETURNS
(SET<VERTEX<person>>) {

  Start = {p};

  Result = SELECT tgt
           FROM Start-(friendship:e)->person:tgt;

  RETURN Result;
}


CREATE QUERY main3(VERTEX<person> p) FOR GRAPH social{
  int cnt = 0;
  SetAccum<VERTEX<person>> @@result;

  Start = {p};

  #select 1-hop neighbor, meanwhile compute 2-hop neighbor
  FirstNeighbors = SELECT tgt
                   FROM Start:s-(friendship:e)->person:tgt
                   ACCUM @@result += one_step(tgt);


  SecondNeighbors = {@@result};
  Result = SecondNeighbors MINUS FirstNeighbors;
  #take the seed out
  Result = Result MINUS Start;

  PRINT Result;
}

INSTALL QUERY all
RUN QUERY main3("Tom")
```