

TigerGraph GSQL Query Language 1.1 Reference Card

Create | Install | Run | Show | Drop Query

```
CREATE [OR REPLACE] QUERY queryName([paramType p1[= defaultVal],...])
FOR GRAPH graphName [RETURNS (returnType)] [API (verId)] {
  [Tuple Definitions] Ⓐ
  [Accumulator Declarations] Ⓑ
  [baseType Declarations]
  [Exception Declarations]
  Query-body Statements Ⓒ
}
```

```
DROP QUERY queryName | ALL | *
INSTALL QUERY queryName | ALL | *
SHOW QUERY queryName
RUN QUERY queryName(parameterValues)
```

Types and Tuple Definition

baseType:
 INT UINT FLOAT DOUBLE STRING
 DATETIME BOOL
 VERTEX<*vTypeName*>
 EDGE<*eTypeName*>
 JSONOBJECT JSONARRAY

paramType:
 baseType
 (except EDGE, JSONOBJECT, JSONARRAY)
 SET<baseType>
 BAG<baseType>

elementType:
 baseType | STRING COMPRESS | *tupleName*

accumType:
 SumAccum<INT|FLOAT|DOUBLE|STRING> AvgAccum
 MaxAccum<INT|FLOAT|DOUBLE>
 MinAccum<INT|FLOAT|DOUBLE>
 OrAccum BitwiseOrAccum
 AndAccum BitwiseAndAccum
 ListAccum<elementType|ListAccum>
 SetAccum<elementType>
 BagAccum<elementType>
 MapAccum<elementType, elementType|accumType>
 ArrayAccum<accumType>
 HeapAccum<tupleName>(size, fieldName ASC|DESC ,...)
 GroupByAccum<elementType aliasName,..., accumType aliasName,... >

Nested accumulator rules:

1. ListAccum: can be nested within ListAccum, up to a depth of 3:
2. MapAccum: All accumulator types, except for HeapAccum, can be nested within MapAccum as the value type.
3. GroupByAccum: All accumulator types, except for HeapAccum, can be nested within GroupByAccum as the accumulator type.

[Ⓐ] **Tuple definition:** TYPEDEF TUPLE < baseType fieldName, ... > tupleName

[Ⓒ] **Statements** In general, a Statement is any of the categories below: Declaration, Accumulator Assignment, Control Flow, DML, or Output. 2 contexts for statements: Query-body level (end with semicolon); DML-sublevel (comma-separated). See GSQL Language Reference for more details.

Ⓑ Declaration Statements

- Declarations must be in order shown in CREATE QUERY syntax.
- At the DML-sublevel, only baseType local variables can be declared.

Global accumulator:

```
[STATIC] accumType<elementType> @@accumName;
```

Vertex-attached accumulator:

```
accumType<elementType> @accumName;
```

Base type:

```
baseType varName;
```

Exception:

```
EXCEPTION exceptVarName "(" errorInt ")";
// errorInt > 40000
```

Vertex set:

```
Below are all ways to declare a seed set.
SO [ (vtype) ] = CREATE QUERY seedSetExample(
  VERTEX<person> v1, SET<VERTEX> v2)
FOR GRAPH gName {
  SetAccum<VERTEX> @@testSet;
  S1 = {v1};
  S2 = v2;
  S3 = @@testSet;
  S4 = ANY; // All vertices
  S5 = person.*; // All person vertices
  S6 = _; // Equivalent to S4
  S7 = S1;
  S9 = S1 UNION S2; // Union of vertex set vars
  S8 = {@@testSet, v1, v2}; // Union of other
vertex variables
}
```

Output Statements

```
printExpr: expr [AS key]
```

PRINT: Output arguments to console in JSON format or to filePath in CSV format if condition is true. Query-body level only.
 PRINT printExpr,... [WHERE condition][TO_CSV filePath];

LOG: Write arguments to GPE log if condition is true. Query-body level or DML-sublevel.
 LOG (condition, printExpr,...);

RETURN: Create a subquery. Return type can be any baseType or accumType, except GroupByAccum or an accumulator type using tuple as the element type.
 CREATE QUERY subQueryName(...)... RETURNS (returnType) {
 ... // query body
 RETURN returnValue; }

Accumulator Assignment Statements

Query-body level or DML-sublevel. Often in ACCUM or POST-ACCUM clause.

```
v.@accumName = expr
v.@accumName += expr // Accumulation
@@accumName = expr // Not allowed at DML-sublevel
@@accumName += expr // Accumulation
```

Exception Statements

Raise statement:

```
RAISE exceptVarName [errorMsg]
```

Try Block:

```
TRY queryBodyStmts
EXCEPTION
  [WHEN exceptVarName THEN queryBodyStmts ]+
  [ELSE queryBodyStmts]
END;
```

DML Statements

Edge-induced SELECT: Only SELECT and FROM are required

```
vSetVarName =  
  SELECT t // vertex alias (s or t)  
  FROM vSetVarName:s - ((eType1|eType2):e) -> (vType1|vType2):t // s,e,t are aliases  
  WHERE condition // Evaluates before ACCUM and POST-ACCUM  
  SAMPLE expr EDGE|TARGET WHEN condition  
  ACCUM DMLSubStatements // Executed on every edge. s, e, and t can all be used.  
  POST-ACCUM DMLSubStatements // 1. If POST-ACCUM is used with ACCUM, the statements follow the  
  // result of ACCUM.  
  // 2. Each POST-ACCUM statement can use only s or only t.  
  HAVING condition // Similar to WHERE, but evaluates after ACCUM and POST-ACCUM  
  ORDER BY expr ASC|DESC, expr ASC|DESC,...  
  LIMIT expr OFFSET expr; // OFFSET is optionally with LIMIT
```

Vertex-induced SELECT: (Supports all the clauses of Edge-induced SELECT. Only the difference from edge-induced SELECT is shown below.)

```
vSetVarName =  
  SELECT s // vertex alias (only s)  
  FROM vSetVarName:s // No edge or target vertex  
  ACCUM DMLSubStatements; // Executed on every vertex.
```

query-body DELETE: delete vertices or edges

```
DELETE aliasName  
FROM vSetVarName:s - (eType1:e) -> (vType1):t  
 // or vSetVarName:s  
WHERE condition;
```

DML-sub DELETE: delete vertices or edges

```
DELETE ( aliasName )
```

INSERT INTO: Insert vertices or edges. Either query-body or DML-sublevel

```
INSERT INTO edgeTypeName (FROM, TO, attr1, attr2)  
VALUES (fromVertexId fromVertexType, toVertexId  
 toVertexType, attrValue1, attrValue2,...);
```

UPDATE: Update vertex or edge attributes

```
UPDATE aliasName  
FROM vSetVarName:s - (eType1:e) -> (vType1):t  
 // or vSetVarName:s  
SET DMLSubStatements  
WHERE condition;
```

Control Flow Statement: can be query-body-level or DML-sublevel

IF statement:

```
IF condition THEN statements  
[ELSE IF condition THEN statements]..  
[ELSE statements] END
```

WHILE statement: (inner statements include CONTINUE BREAK)

```
WHILE condition [LIMIT intExpr]  
DO statements END
```

FOREACH statement: (inner statements may include CONTINUE or BREAK)

```
FOREACH varName IN setBagExpr DO statements END  
FOREACH varName IN RANGE [ expr, expr ].STEP( expr ) DO statements END
```

CASE statement: Trigger ONLY the first statements whose condition is true.

```
CASE [WHEN condition THEN statements]+ ELSE statements END  
CASE expr [WHEN constant THEN statements]+ ELSE statements END
```

Operators

Math operators: + - * / % << >> & |

Comparison operators: < <= > >= == !=

String operator: +

Boolean operators: NOT AND OR

Boolean constant: TRUE FALSE

Other operators for condition:

```
expr BETWEEN expr AND expr  
expr [NOT] LIKE expr  
expr IS [NOT] NULL
```

Set|Bag operators:

```
setBagExpr UNION | INTERSECT | MINUS setBagExpr  
expr [NOT] IN setBagExpr
```

Collections

```
(1, 2) // a set or bag  
("a" -> 2) // key-value pair for map  
["abc", "def"] // a list
```

Built-in Functions

Categories of Built-in Functions

See GSQL Language Reference for full list

Math functions

Type conversion functions

DATETIME functions

JSONARRAY and JSONOBJECT parsing functions

VERTEX functions:

```
INT v.outdegree( [STRING] )  
BAG<VERTEX> v.neighbors( [STRING] )  
BAG<attr> v.neighborAttributes(STRING,STRING,STRING)  
BAG<attr> v.edgeAttribute(STRING, STRING)
```

EDGE function:

```
BOOL e.isDirected()
```

Aggregation functions: The argument is a set or bag

```
COUNT SUM MIN MAX AVG
```