TigerGraph 3.1 GSQL Query Language Reference Card

CREATE | INTERPRET | SHOW | RUN | INSTALL | DROP QUERY

```
CREATE [OR REPLACE] [DISTRIBUTED] QUERY queryName([paramType
p1[= defaultVal],…])
[FOR GRAPH graphName] [RETURNS (returnType)][API
verID][SYNTAX (verId)]
{
        [Tuple Definitions]
        [baseType,Accumulator,fileType Declarations]
        [Exception Declarations]
        Query-body Statements
}
```

```
INTERPRET QUERY ()
[for graphName]
[SYNTAX verID]
{
   [Tuple Definitions]
   [baseType,Accumulator,fileType

   Declarations]

   [Exception Declarations]

   Query-body Statements
}
```

```
INSTALL QUERY [options] queryName | ALL | *
   options:
   -FORCE
   -DISTRIBUTED
```

```
RUN QUERY [runOptions] queryName(parameters)
   runOptions:
   -av
   -async

DROP QUERY queryName | ALL | *

SHOW QUERY queryName
```

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>
```

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

Declaration statements

• Declarations must be in the order shown in CREATE QUERY syntax.
• At the DML-sub level, only base type local variables can be declared.

Global accumulator:
```
[STATIC] accumType<elementType> @@accumName;
```

Vertex-attached accumulator:
```
accumType<elementType> @accumName;
```

Base type:
```
baseType varName [=initValue];
```

File type:
```
FILE fileVar "("filePath")";
```

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

Vertex set:
```
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 statement:
```
PRINT printExpr,… [WHERE condition]
[TO_CSV {filePath|fileVar}];
```

println:
```
fileVar".println (" expr,…")";
```

LOG statement:
```
LOG (condition, printExpr,…);
```

RETURN statement: Used in subqueries only.
```
CREATE QUERY subQueryName(…)… RETURNS (returnType) {
… // query body
RETURN returnValue; }
```

Accumulator Assignment Statements

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

```
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

SELECT statement

SYNTAX V1:
```
vSetVarName =
SELECT t // vertex alias (s or t)
FROM vSetVarName:s – ((eType1|eType2):e) - vType:t //
s,e,t are aliases WHERE condition
WHERE condition // Evaluates before ACCUM and
POST-ACCUM
SAMPLE expr EDGE|TARGET WHEN condition
ACCUM DMLSubStatements
POST-ACCUM DMLSubStatements
// Executed on every edge. s, e, and t can all be
used.
// 1. If POST-ACCUM is used with ACCUM, the statements
follow the // result of ACCUM.
```

SYNTAX V2:
```
vSetVarName =
SELECT s // vertex alias (s or t)
FROM vType1:s - (<eType1.<eType2.eType3>) - vType2:t
// Source set is treated the same as target - no
longer need to declare seed set
WHERE condition // Evaluates before ACCUM and
POST-ACCUM
SAMPLE expr EDGE|TARGET WHEN condition
PER s // Optional clause that affects the execution of
the ACCUM clause
ACCUM DMLSubStatements // Executed on every edge
unless a PER clause limits its scope. s, e, and t can
all be used.
POST-ACCUM DMLSubStatements
```

```
// 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,...
ORDER BY expr ASC|DESC, expr ASC|DESC,...
LIMIT expr OFFSET expr; // OFFSET is optionally with LIMIT
```

```
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.
                             // 3. In Syntax V2, one SELECT statement can have multiple POST-ACCUM clauses
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 optional with LIMIT
```

SQL-Like SELECT Statement:

```
SELECT s.attribute, t.attribute, s2 … INTO table
FROM vType1:s - ((eType1>|<eType2):e) - vType2:s2 - (_*..2) - (vType3):t
WHERE condition
GROUP BY groupExpr, groupExpr
HAVING condition
ORDER BY expr ASC|DESC, expr ASC|DESC,…
LIMIT ( expr,expr OFFSET expr )
```

Query-body DELETE:

```
DELETE aliasName

FROM vSetVarName:s – (eType1:e) -> (vType1):t

// or vSetVarName:s

WHERE condition;
```

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,…);
```

DML-sub DELETE: delete vertices or edges

```
DELETE ( aliasName )
```

UPDATE: Update vertex or edge attributes

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

Control Flow Statements

IF statement:

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

WHILE statement:

```
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, Functions, and Expressions

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

Set | Bag: `(1, 2)`

Key-value pair for map: `("a" -> 2)`

List: `["abc", "def"]`

Built-in functions categories

Math functions

String 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 functions:

`BOOL e.isDirected()`

Aggregation functions: The argument is a set or bag

`COUNT SUM MIN MAX AVG`