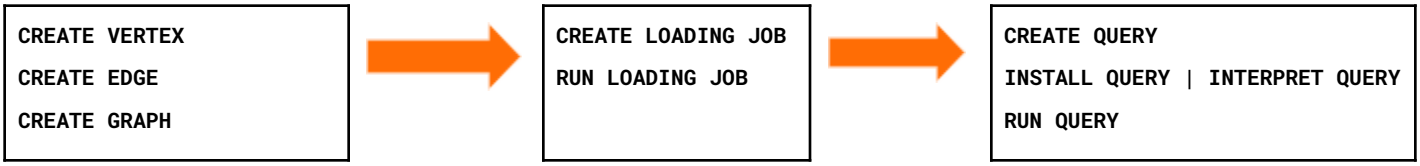
TigerGraph GSQL DDL Language Reference Card

Workflow:



Define a schema

```
DROP ALL # erases all graph and job definitions, and clears graph store

CREATE VERTEX vertexType (attributeName type PRIMARY KEY, [attributeName type [DEFAULT defaultValue]]*) [ WITH
vertexOption (,vertexOption)* ]

vertexOption:
   TAGGABLE=("TRUE" | "FALSE")
   STATS=("outdegree_by_edgetype" | "none" )
   primary_id_as_attribute=("TRUE" | "FALSE")

CREATE UNDIRECTED EDGE edgeType (FROM vertexType, TO vertexType [, attributeName type [DEFAULT defaultValue]]*
)

CREATE DIRECTED EDGE edgeType (FROM vertexType, TO vertexType [, attributeName type [DEFAULT defaultValue]]* )
[WITH REVERSE_EDGE="reverseEdgeType"]

CREATE GRAPH graphName [AS baseGraphName ] (vertex_or_edge_type[:tag], vertex_or_edge_type[:tag]...)
[WITH ADMIN username]

USE GRAPH graphName # set graphName to be the active graph

USE GLOBAL

DROP GRAPH graphName

TYPEDEF TUPLE<f1 INT(b), f2 UINT, f4 STRING(n)> tupleName # Create a UDT type on the catalog level
```

Attribute types

type: INT | UINT | FLOAT | DOUBLE | BOOL | STRING | STRING COMPRESS (deprecated) | FIXED_BINARY(n) | DATETIME | UDT | LIST<elementType> | SET<elementType> | MAP<keyType, valueType>

UDT: TYPEDEF TUPLE<f1 INT(b), f2 UINT, f4 STRING(n)> tupleName

LIST|SET element and MAP value type: INT, DOUBLE, STRING, STRING COMPRESS, DATETIME, UDT

MAP keyType: INT, STRING, STRING COMPRESS, DATETIME

Schema Change – Modify Vertex/Edge Types

```
CREATE [GLOBAL] SCHEMA_CHANGE JOB jobName FOR GRAPH graphName {
[sequence of DROP, ALTER, and ADD statements, each line ending with a semicolon]
}

RUN [GLOBAL] SCHEMA_CHANGE JOB jobName;
```

```
ADD VERTEX vertexType (PRIMARY_ID id type …);  // same syntax as CREATE VERTEX

ADD UNDIRECTED EDGE edgeType (FROM vertexType…); // same syntax as CREATE UNDIRECTED EDGE

ADD DIRECTED EDGE edgeType (FROM vertexType… );  // same syntax as CREATE DIRECTED EDGE

ALTER VERTEX|EDGE objectTypeName ADD|DROP (attributeList);

ALTER VERTEX vertexType ADD INDEX indexTypeName ON (attributeName);

ALTER VERTEX vertexType WITH TAGGABLE = (“TRUE” | “FALSE”);

DROP VERTEX vertexType [, vertexType]*;

DROP EDGE edgeType [, edgeType]*;

ADD TAG tagName [ DESCRIPTION description ];

DROP TAG tagName [ DESCRIPTION description ];
```

Vertex Level Access Control

Workflow:

Define tags and mark vertex types as taggable:

```
CREATE SCHEMA_CHANGE JOB{

        ADD TAG vip DESCRIPTION “very important person”

        ALTER VERTEX person WITH TAGGABLE="true"
}
```

Create tag-based graphs

```
USE GRAPH socialNet

CREATE GRAPH vip AS socialNet (person:vip, post, friend, posted, liked)
```

Assign roles to users

```
GRANT ROLE admin [ON GRAPH vip ] TO user1,... userN

REVOKE ROLE queryWriter [ON GRAPH vip ] FROM user1,... userN
```

Tag vertices

Use TAGS () BY in LOAD statements

Load data into tag-based graphs

Use tag functions to add tags to existing data

Create a LOADING JOB

```
CREATE LOADING JOB jobName FOR GRAPH gname {
[zero or more DEFINE statements]
[zero or more LOAD statements] | [zero or more DELETE statements]
}
```

DEFINE statements:

```
DEFINE FILENAME fileVar [= filePath];
filePath = (path | all: path | any: path | machAliases: path ["," machAliases: path]* )
mach_aliases = list of machine aliases, e.g,, m1,m3
DEFINE HEADER headerName = "columnName"[,"columnName"]*;
DEFINE INPUT_LINE_FILTER filterName = boolean_expression_using_column_variables;
```

LOAD statements:

```
LOAD (fileVar|filepathString|TEMP_TABLE tableName) destinationClause [,destinationClause]*
[TAGS (tag1, tag2, ...) BY (OR|OVERWRITE) ][USING parsingConditions];
```

DELETE statement:

```
DELETE VERTEX vertexType (PRIMARY_ID id_expr) FROM (fileVar|filePath) [WHERE condition];
DELETE EDGE edgeType (FROM id_expr [, TO id_expr]) FROM (fileVar|filePath) [WHERE condition];
DELETE EDGE * (FROM id_expr vertexType) FROM (fileVar|filePath) [WHERE condition];
```

destinationClause:

```
TO VERTEX|EDGE name VALUES (id_expr [,attr_expr]* )[WHERE conditions]
TO TEMP_TABLE name (idName [,attrName]*) VALUES (id_expr [,attr_expr]* )[WHERE conditions]
```

parsingConditions:

```
parameter=value [parameter=value]*
SEPARATOR=sChar HEADER="true"|"false"
EOL=eChar
QUOTE="single"|"double" USER_DEFINED_HEADER="true"|"false"
REJECT_LINE_RULE=filterName JSON_FILE="true"|"false"
```

```
id_expr: attr_expr|REDUCE(reducer_func_name(attr_expr))
attr_expr: $1|$"columnName"|token_func_name(attr_expr[, attr_expr]* )
attr_expr for UDT: tupleName($1, $2, …)
attr_expr for LIST|SET: $1 | SPLIT($1, “,”)
attr_expr for MAP: $1 -> $2 | SPLIT($1, “,”) | SPLIT($1, “,”, “:”)
token_func_name: see Language Reference - “Creating a Loading Job” - "Built-in Loader Token Functions"
reducer_func_name: max, min, add, and, or
WHERE condition:
Operators: +, -, *, /, <, >, ==, !=, <=, >=, AND, OR, NOT, IS NUMERIC, IS EMPTY, IN, BETWEEN..AND
```

Load Data and Manage Loading Jobs

```
CLEAR GRAPH STORE [-HARD] # erases all graph data. Note: DROP GRAPH & DROP ALL do this automatically.
```

```
RUN LOADING JOB [loading_options] job_name [USING fileVar[=filePath] [, fileVar[=filePath]]* ]


loading_options:
-n [firstLineNum,] lastLineNum
-dryrun
-noprint
SHOW LOADING STATUS jobId|ALL
ABORT LOADING JOB jobId|ALL
RESUME LOADING JOB jobId
```