User-Defined Indexes

TigerGraph Version 3.0

Introduction

User-defined Indexes or Secondary Indexes (as they are commonly called in Database Industry) are a critical feature that enhance the performance of a database system. Indexes allow users to perform fast lookups on non-key columns or attributes without a full-fledged scan. Although this feature is a popular feature for relational databases, it has become essential for Graph databases as well.

Feature Description

Overview

This is a brand-new feature that is being introduced in Ver 3.0. This is primarily a performance enhancement for Querying the data. It is important to note that Index maintenance will result in increased update times as well as additional disk space.

GSQL language has extended DDL, IUD and Query commands to support User-defined Indexes. The new syntax is based on existing commands to ensure similarity and continuity.

High-level Design

TigerGraph Database allows users flexibility to define User-defined Indexes on Attributes. The following table provides details on supported operations for Indexes:

Command	Statement	Supported Functionality
DDL	Index creation	User has the flexibility to create Index in an empty graph initially or to add Index later when the database is running. If the index is added on an existing Vertex, Index data is built in the background. Additionally, attribute and Index can be added in the same command.
	Drop Index	Indexes can be dropped anytime. There are no restrictions.
	Data type support	Only the following data types are supported currently: STRING, UINT, INT, DATETIME, STRING COMPRESS

IUD	Consistency	Strong consistency guarantees are provided for data in Index used for querying
	Transaction	Indexes will continue to support existing TigerGraph transaction semantics
DML	Select Queries	There are no Index-specific changes to syntax. Querying in Distributed and Interpreted mode are supported for Indexes.
	Predicate Type support	The following predicate types are supported in the decision to figure out which indexes to use: Equality, Range, LIKE, IN clause, Logical operators (AND, OR, NOT).
	Function support	Built-In functions and Queries are not supported.
Misc	BAR support	Indexes will be backed up and restored as part of existing BAR functionality. When additional nodes are added to a cluster, the indexes get scaled automatically.
	Distributed Mode	There is no distinction with single server mode.

Syntax

DDL:

ALTER VERTEX statement is used to add to or remove Indexes from an existing vertex type. It may only be used within a SCHEMA_CHANGE JOB. The basic format is as follows:

```
create global schema_change job <job-name>
{
  alter vertex <vertex-name> add index <index-name> on (<attr-name>);
  alter vertex <vertex-name> drop index <index-name>;
};
```

DML/IUD

Indexes will be supported for all DML/IUD commands with some exceptions.

The following type of predicates are supported:

- Equality Predicates
  - select <sel-list> from <vertex-name> where <attr-name> = <val1>;
- IN Clause
  - select <sel-list> from <vertex-name> where <attr-name> in ('<val1>, <val2>);
- Range Predicates
  - select <sel-list> from <vertex-name> where <attr-name> > <val1>;
  - select <sel-list> from <vertex-name> where <attr-name> < <val1> and <attr-name> >= <val2>;
- LIKE Predicate
  - select <sel-list> from <vertex-name> where <attr-name> like '<string%>'

Motivation

Indexes allow users to perform fast looks on non-primary key attributes. Without indexes, queries on non-key attributes would need to read all the data in the graph. So, Indexes are very important for data retrieval performance. However, adding indexes will decrease write performance. For this reason, users should be judicious about adding indexes. Users should review the querying patterns to decide where Indexes can help.

Use Cases/Examples

- Simple use case to add an Index on an existing Vertex and retrieve data

DDL:

```
> show Vertex person;
CREATE VERTEX person ( PRIMARY_ID personId STRING,
```

```
                id STRING,
                locationId STRING,
                skillSet SET<INT>,
                skillList LIST<INT>,
                interestSet SET<STRING COMPRESS>,
                interestList LIST<STRING COMPRESS>);

CREATE GLOBAL SCHEMA_CHANGE JOB Ind_test
{
ALTER VERTEX person ADD INDEX Person_Ind ON ( locationId);
ALTER VERTEX person ADD ATTRIBUTE (DateOfBirth int);
ALTER VERTEX person ADD INDEX Person_DoB ON (DateOfBirth);
}
```

DML:

There is no special syntax to invoke use of Indexes. Examples:

Equality:

```
SELECT s
FROM va - (ANY) - vb
WHERE va.x == 1
```

IN

```
SELECT s
FROM va - (ANY) - vb
WHERE va.y in ('abc', 'def')
```

Range (Both 'Compare' and 'Between' Conditions)

```
SELECT s
FROM va - (ANY) - vb
WHERE va.z > 10
```

or

```
SELECT s
FROM va - (ANY) - vb
WHERE va.z>= 1 and va.z , 10
```

String prefix LIKE

```
SELECT s
FROM va - (ANY) - vb
WHERE va.s like'abc%'
```

Additional considerations:

Support for Queries with Conjunctive normal form

Use flexibility to use indexes on subset of columns in the predicate list.

Example: Use Index on 'x', or both indexes on 'y' AND 'z' in the following predicate

```
x = 1 and (y in ('abc', 'def') or z > 10)
```

NOT clause

Index is use is also considered when NOT clause evaluated as part of all supported query predicates.

```
x = 1 and (y NOT in ('abc', 'def'))
```