Apache TinkerPop

Apache TinkerPop™

Apache TinkerPop™ is a graph computing framework for both graph databases (OLTP) and graph analytic systems (OLAP).

Gremlin Query Language

Gremlin is the graph traversal language of Apache TinkerPop. Gremlin is a functional, data-flow language that enables users to succinctly express complex traversals on (or queries of) their application's property graph. Every Gremlin traversal is composed of a sequence of (potentially nested) steps. A step performs an atomic operation on the data stream. Every step is either a map-step (transforming the objects in the stream), a filter-step (removing objects from the stream), or a sideEffect-step (computing statistics about the stream). The Gremlin step library extends on these 3-fundamental operations to provide

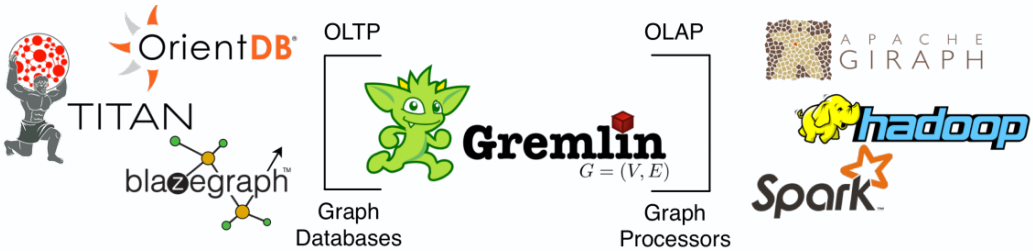
Apache TinkerPop

```
g.V().has("name","gremlin").as("a").
  out("created").in("created").
    where(neq("a")).
  groupCount().by("title")
```

Get the distribution of titles amongst Gremlin's collaborators.

1. Get the vertex with the name "gremlin" and label it "a."
2. Get Gremlin's created projects and then who created them...
3. ...that are not Gremlin.
4. Group count those collaborators by their titles.

1 2 3 4 5



OLTP and OLAP Traversals

evaluated as either a real-time database query or as a batch analytics query. The former is known as an online transactional process (OLTP) and the latter as an online analytics process (OLAP). This universality is made possible by the Gremlin traversal machine. This distributed, graph-based virtual machine understands how to coordinate the execution of a multi-machine graph traversal. Moreover, not only can the execution either be OLTP or OLAP, it is also possible for certain subsets of a traversal to execute OLTP while others via OLAP. The benefit is that the user does not need to learn both a database query language and a domain-specific BigData analytics language (e.g. Spark DSL, MapReduce, etc.). Gremlin is all that is required to build a graph-based application because the Gremlin traversal machine will handle the rest.

```
g.V().has("name","gremlin").as("a").
  out("created").in("created").
  where(neq("a")).
  in("manages").
  groupCount().by("name")
```

Imperative & Declarative Traversals

A Gremlin traversal can be written in either an imperative (procedural) manner, a declarative (descriptive) manner, or in a hybrid manner containing both imperative and declarative aspects. An imperative Gremlin traversal tells the traversers how to proceed at each step in the traversal. For instance, the imperative traversal in the first box first places a traverser at the vertex denoting Gremlin. That traverser then splits itself across all of Gremlin's collaborators that are not Gremlin himself. Next, the traversers walk to the managers of those collaborators to ultimately be grouped into a manager name count distribution. This traversal is imperative in that it tells the traversers to "go here and then go there" in an explicit, procedural manner.

```
g.V().match(
  as("a").has("name","gremlin"),
  as("a").out("created").as("b"),
  as("b").in("created").as("c"),
  as("c").in("manages").as("d"),
   where("a",neq("c"))).
```

A declarative Gremlin traversal does not tell the traversers the order in which to execute their walk, but instead, allows each traverser to select a pattern to execute from a collection of (potentially nested) patterns. The declarative traversal in the second box yields the same result as the imperative traversal above. However, the declarative traversal has the added benefit that it leverages not only a compile-time query planner (like imperative traversals), but also a runtime query planner that chooses which traversal pattern to execute next based on the historic statistics of each pattern -- favoring those patterns which tend to reduce/filter the most data.

The user can write their traversals in any way they choose. However, ultimately when their traversal is compiled, and depending on the underlying execution engine (i.e. an OLTP graph database or an OLAP graph processor), the user's traversal is rewritten by a set of traversal strategies which do their best to determine the most optimal execution plan based on an understanding of graph data access costs as well as the underlying data systems's unique capabilities (e.g. fetch the Gremlin vertex from the graph database's "name"-index). Gremlin has been designed to give users flexibility in how they express their queries and graph system providers flexibility in how to efficiently evaluate traversals against their TinkerPop-enabled data system.

Host Language Embedding

databases require the developer to code both in their native programming language as well as in the database's respective query language. An argument can be made that the difference between "query languages" and "programming languages" are not as great as we are taught to believe. Gremlin unifies this divide because traversals can be written in any programming language that supports function composition and nesting (which every major programming language supports). In this way, the user's Gremlin traversals are written along side their application code and benefit from the advantages afforded by the host language and its tooling (e.g. type checking, syntax highlighting, dot completion, etc.). Various Gremlin language variants exist including: Gremlin-Java, Gremlin-Groovy, Gremlin-Python, Gremlin-Scala, etc.

The first example below shows a simple Java class. Note that the Gremlin traversal is expressed in Gremlin-Java and thus, is part of the user's application code. There is no need for the developer to create a String representation of their query in (yet) another language to ultimately pass that String to the graph computing system and be returned a result set. Instead, traversals are embedded in the user's host programming language and are on equal footing with all other application code. With Gremlin, users do not have to deal with the awkwardness exemplified in the second example below which is a common anti-pattern found throughout the industry.

```
public Class GremlinTinkerPopExample{
 public void run(String name, String property){
  Graph graph = GraphFactory.open(...);
  GraphTraversalSource g = traversal().withEmbedded(graph);

  double avg = g.V().has("name",name).
         out("knows").out("created").
         values(property).mean().next();

   System.out.printIn("Average rating :" +avg);
  }
 }
```

```
public Class Sql3dbcExample {
 public void run(String name, String property){
  Connection connection = DriverManager.getConnection(...)
```

Apache TinkerPop

```
p1" +
   "INNER JOIN KNOWS K ON k. person1 pl.id +
   "INNER JOIN PERSONS P2 ON p2.id k.person2 +
   "INNER JOIN CREATED C ON c.person = p2.id " +
   "INNER JOIN PROJECTS Pr ON pr.id c.project
   "WHERE p.name " + name + "');
   System.out.println("Average
rating"+result.next().getDouble("AVERAGE"));
  }
 }
```

Behind the scenes, a Gremlin traversal will evaluate locally against an embedded graph database, serialize itself across the network to a remote graph database, or send itself to an OLAP processor for cluster-wide distributed execution. The traversal source definition determines where the traversal executes. Once a traversal source is defined it can be used over and over again in a manner analogous to a database connection. The ultimate effect is that the user "feels" that their data and their traversals are all co-located in their application and accessible via their application's native programming language. The "query language/programming language"-divide is bridged by Gremlin.

```
Graph graph = GraphFactory.open(...);
GraphTraversalSource g;
g = traversal().withEmbedded(graph);          //local OLTP
g =
traversal().withRemote(DriverRemoteConnection.using("Localho
st",8182));      //remote
g =
traversal().withEmbedded(graph).withComputer(SparkGraphCompu
ter.class);     //distributed OLAP
```

Related Resources

Apache TinkerPop

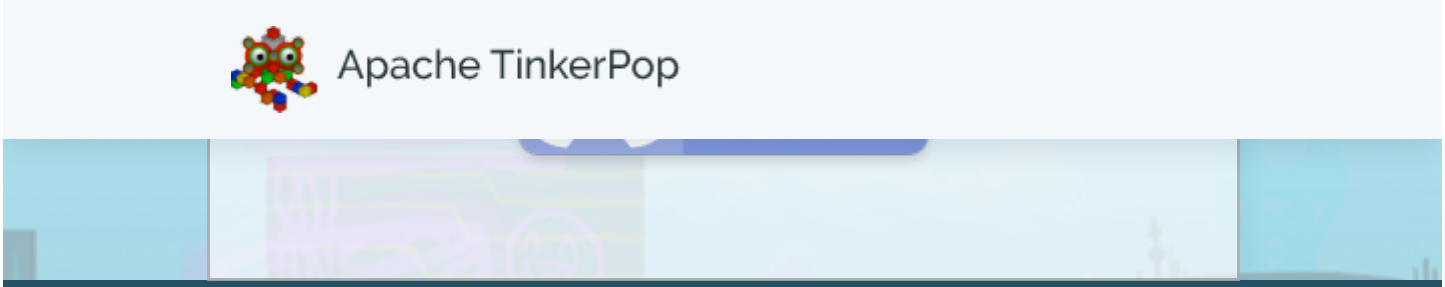
The Gremlin Graph Traversal Machine and Language

SQL2Gremlin Introduction

How to Succeed with Apache Cassandra

Join Us

Join us on our Discord Server



Apache TinkerPop™

Apache TinkerPop™ is a graph computing framework for both graph databases (OLTP) and graph analytic systems (OLAP).

Links