OrientDB: Using Schemas with Graphs, Part 1

OrientDB is a Graph Database "on steroids" because it supports concepts taken from both the Document Database and Object-Oriented worlds.

Take a look at this use case: Creating a graph to map the relationships between Person and Cars. We're going to use the just-released OrientDB version 1.5.

Let's open a shell (or command prompt in Windows) and launch the OrientDB Console (use console.bat on Windows):

```
> cd $ORIENTDB_HOME/bin
> ./console.sh
```

Now we're going to use the console to create a brand new local database:

```
orientdb> create database plocal:../databases/cars admin admin plocal graph
```

Ok, now let's create the first graph schema with "Person" and "Car" as 2 new Vertex types and "Owns" as an Edge type:

```
orientdb> create class Person extends V
orientdb> create class Car extends V
orientdb> create class Owns extends E
```

And let's go populate the database with the first Graph elements:

```
orientdb> create vertex Person set name = 'Luca'

Created vertex 'Person#11:0{name:Luca} v1' in 0,012000 sec(s).

orientdb> create vertex Car set name = 'Ferrari Modena'

Created vertex 'Car#12:0{name:Ferrari Modena} v1' in 0,001000 sec(s).

orientdb> create edge Owns from (select from Person) to (select from Car)

Created edge '[e[#11:0->#12:0][#11:0-Owns->#12:0]]' in 0,005000 sec(s).
```

Ok, now we can traverse vertices. For example, what is Luca's car? Traverse from Luca vertex to the outgoing vertices following the "Owns" relationships:

```
orientdb> select name from ( select expand( out('Owns') ) from Person where name = 'Luca' )

----+-----+--------------
#   |@RID  |name
----+-----+--------------
0   |#-2:1 |Ferrari Modena
----+-----+--------------
```

Perfect.

Now we have the location of Person and we need another Vertex type called "Country" to connect to the person with a new "Lives" Edge type:

```
orientdb> create class Country extends V
orientdb> create class Lives extends E

orientdb> create vertex Country set name = 'UK'

Created vertex 'Country#14:0{name:UK} v1' in 0,004000 sec(s).
```

Next, let's associate Luca to the UK Country:

```
orientdb> create edge Lives from (select from Person) to (select from Country)

Created edge '[e[#11:0->#14:0][#11:0-Lives->#14:0]]' in 0,006000 sec(s).
```

So far so good. Our graph has been extended.
Now, try to search the country where there are "Ferrari" cars in our database.
orientdb> select name from (select expand(in('Owns').out('Lives')) from Car where name like '%Ferrari%')

—-+—–+—-
|@RID |name
—-+—–+—-
0 |#-2:1|UK
—-+—–+—-

Setting constraints on Edges
Now we've modeled our graph using a schema without any constraints. But it would be useful to require an Owns relationship to exist only between the Person and Car vertices. So, let's create these constraints:

orientdb> create property Owns.out LINK Person
orientdb> create property Owns.in LINK Car
The MANDATORY setting against a property prevents OrientDB from using a lightweight edge (no physical document is created). Be sure to pay attention and not put spaces between MANDATORY=true.

orientdb> alter property Owns.out MANDATORY=true;
orientdb> alter property Owns.in MANDATORY=true;

If we want to prohibit a Person vertex from having 2 edges against the same Car vertex, we have to define a UNIQUE index against out and in properties.

orientdb> create index UniqueOwns on Owns(out,in) unique

Created index successfully with 0 entries in 0,023000 sec(s).

Unfortunately, the index tells us 0 entries are indexed. Why? We have already created the Owns relationships between "Luca" and "Ferrari Modena." In that case, OrientDB had already created a lightweight edge before we set the rule to force creation documents for Owns instances. So, you need to drop and recreate the edge.

orientdb> delete edge from #11:0 to #12:0
orientdb> create edge Owns from (select from Person) to (select from Car)

Now check that the record has been created.

orientdb> select from Owns

—-+—–+—–+—–
|@RID |out |in
—-+—–+—–+—–
0 |#13:0|#11:0|#12:0

—-+—–+—–+—–

So far so good. The constraints works. Now try to create a "Owns" edge between Luca and UK (Country vertex):
orientdb> create edge Owns from (select from Person) to (select from Country)

Error: com.orientechnologies.orient.core.exception.OCommandExecutionException: Error on execution of command: sql.create edge Owns from (select from Person) to (sel…
Error: com.orientechnologies.orient.core.exception.OValidationException: The field 'Owns.in' has been declared as LINK of type 'Car' but the value is the document #14:0 of class 'Country'

Now we have a typed graph with constraints.

The next part will cover how to use polymorphism feature in the graph.

Luca Garulli, CEO
Orient Technologies, the Company behind OrientDB
http://about.me/luca.garulli

One response on "*OrientDB: Using Schemas with Graphs, Part 1*"

Luca Garulli says:

August 10, 2013 at 3:22 pm

Log in to Reply

For more information look at: http://www.orientechnologies.com/docs/last/orientdb.wiki/Graph-Schema.html

Leave a Reply

You must be logged in to post a comment.

About us

OrientDB Ltd is a private company that officially leads the development of the OrientDB Open Source Project. The original author and the main committers of the project created the company in 2011 to provide Enterprise Support for such technology.

Latest News

Released OrientDB 2.1.12
March 2, 2016

Stay Tuned For Our OrientDB v2.2 Beta Announcement!
February 17, 2016

Released OrientDB 2.1.11
February 15, 2016

OrientDB v2.2 exposes storage performance metrics through JMX – By Andrey Lomakin
February 8, 2016

Released OrientDB 2.1.10
February 4, 2016

Subscribe to Newsletter

Enter your business email to subscribe

Submit