Getting Started with OrientDB

In my previous blog post I praised the open-source database Orient DB. I've received many email from people since asking me questions about Orient DB or telling me that they'd love to try Orient DB out but don't know where to start.

I though it is time for me to contribute to the Orient DB community and write a short tutorial for exactly this audience. Ideally, I'd like to ensure that if you sit through the videos below, you'd at least have an informed view of whether Orient DB is of interest to you, your project and your organization.

Think of this blog post as the first installment. I may come back with some more advanced tutorials later. This tutorial shows you how to:

- Download and install Orient DB
- Play with the Orient DB studio and perform queries on the Grateful Dead database provided with the community edition
- Create your own database (again using Orient DB studio, perhaps later I'll provide a tutorial to show how to do this from Node or Java)
- Create server-side functions
- Perform graph- and document-based queries over Orient DB

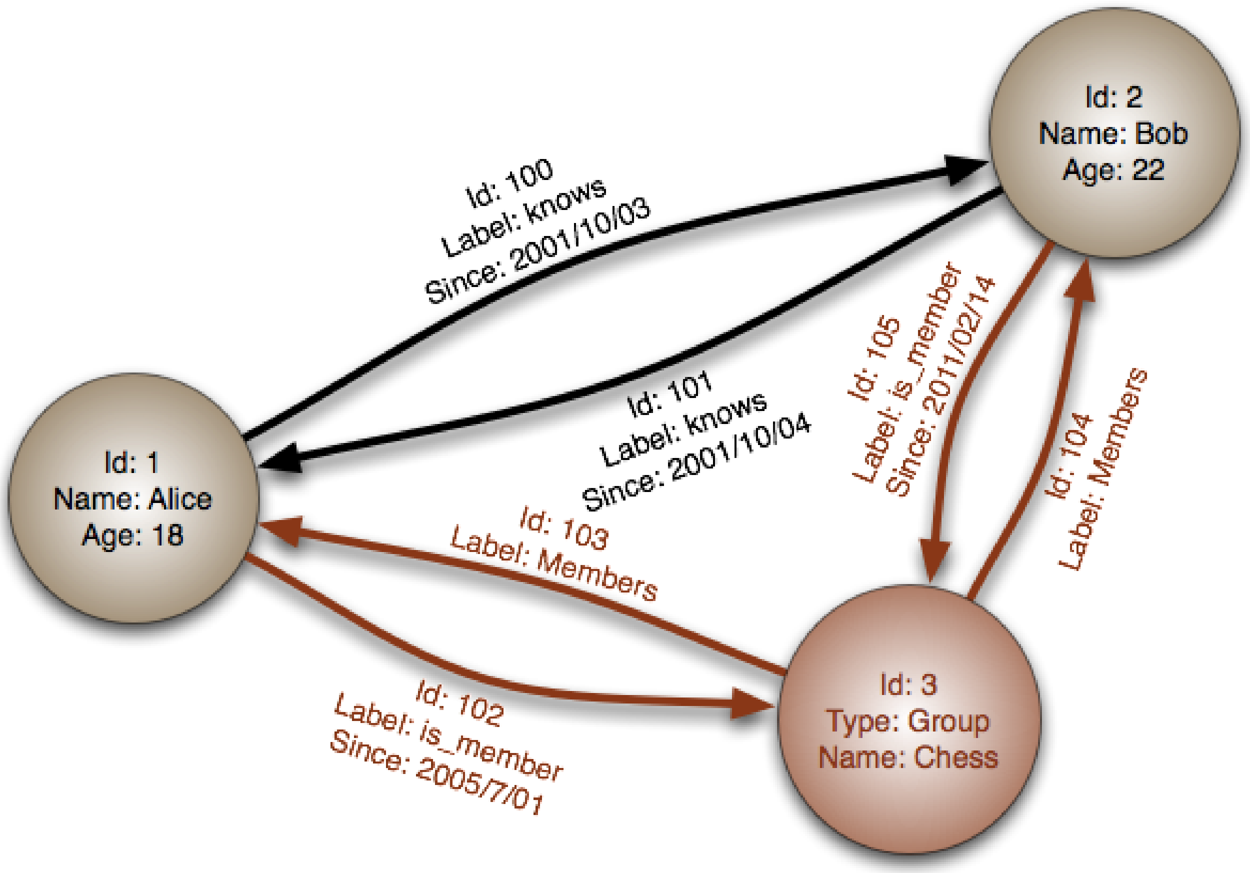
I hope this tutorial gives you a good idea of how Orient DB works and some of its power. I also hope it will help illustrate the reasons for my enthusiasm in the previous post [http://www.scispike.com/orientdb-business-database] where I argued the case for Orient DB. In this post I'll focus on how you get started.

Introduction (or, Read Me First!)

I had some of my guys review the videos for this tutorial and they pointed out that I assumed that the viewer/reader knew something about graphs. Although this may be a bad assumption, I don't want to patronize you.

This is my attempt at giving you a short introduction to the concepts of graphs and the vocabulary I'll use in the tutorial.

Orient DB is not only a graph database. It supports all (or nearly all) the features of a document database and an object-oriented database. Hence, discussing only graph theory may be a bit misleading. However, since I'm using primarily the Graph API of Orient DB in this tutorial, it may be prudent of me to ensure you know something about graphs.
Here is what you need to know.



The above picture is taken from the Wikipedia article on Graph Databases [http://en.wikipedia.org/wiki/Graph_database] . The illustration shows 3 objects (as circles) that we keep track of. There are also lines (or arrows) that define relationship between the objects. We call an object vertex and a relationship edge.

I do have some issues with the graph. They always show two edges that only differs based on the direction they are read. I would have preferred a single line (or a single edge). However, since it is likely that you would lookup Graph Databases in Wikipedia, I thought it would be better if I showed their example and criticized it (OK, I will go and suggest another drawing for Wikipedia. It is on my to do list. Honestly!)

A graph (for the purpose of this tutorial) consists of:

- Vertices. A vertex is a cluster of information. The information is typically stored in key-value pairs (aka properties). You may think of it as a map (or I've notice that many developers call these maps hash tables, however, I find this misleading as hash is a particular strategy for arranging keys in one of the map implementations).
  One way of seeing a vertex is as an identifiable set of properties, where a property is defined as a key-value pair. Unique to Orient DB, the properties can be nested to form complex data structures. I will not do any nesting in this tutorial. I do, however, plan to write another tutorial where I explore Orient DB as a document database.
- Edges. An edge is a link between two vertices. Edges may also have properties. Edges have have direction. That is, they start in one vertex and end in another vertex (actually, to be correct, they may also end in the same vertex. Such edge has the cute name 'buckle' in graph theory).

That's it! Really.

Most of you probably have some background in relational databases, so let me give you a comparison between OrientDB and typical relational databases. The comparison will be slightly misleading no matter how I present it. To minimize the confusion, I'll decided to add a column to discuss how the two technologies differs also.

Graph DB	**Relational DB**	**Differences**
Vertex	Row	A row in a RDB is a flat set of properties. In Orient DB the vertex may be an arbitrarily complex structure. If you are familiar with document databases, it is basically a document.
Edge	Relationship	A relationship in a relational database is not a first class citizen. The relationship is made ad-hoc by use of joins on keys. In Orient DB (as in all Graph Databases) the edge (relationship) is a first class citizen. This mean it can have an id, have properties, etc.

You may ask, where is the comparison of schema constructs such as table, column, trigger, store-procedure, etc. I think the tutorial does discuss these constructs, so I'll delay the discussion to later in this tutorial.

In the second tutorial, I will repeat this short theory session. To help you progress from this incomplete theory session (and for your reading pleasure), I've provided a few links below:

- A presentation of graph databases and Orient DB on slide-share [http://www.slideshare.net/graphdevroom/works-with-persistent-graphs-using-orientdb]
- A bit of graph theory from Wikipedia (when you see the Greek symbols, simply stop reading, you really don't need to know [http://en.wikipedia.org/wiki/Graph_theory]
- A great YouTube presentation on Orient DB from Luca Garulli [https://www.youtube.com/watch?v=o_7NCiTLVis]
- A YouTube presentation on uses of Graph Databases from Jans Aasman [https://www.youtube.com/watch?v=rcDp05O6Iy0]
- A free download of a book on graph databases (the examples are in Neo4J and the book was written by the Neo4J guys) [http://graphdatabases.com/]

I decided to use Orient DB Studio in the videos. It is probably the least likely tool that you'll use to access the database. Most developers prime exposure to Orient DB is through language bindings or libraries. I was thinking of selecting one of these environments and show how Orient DB is used in there. The problem is that if I picked one of the environments I would risk alienating the ones using other environments. I think no matter what environment you're using, you would at some point would bring up Orient DB studio, but perhaps I'm wrong and I just alienated everyone. I hope not!
If time permits, I'll come back with separate tutorials for how to use Orient DB from Java/Scala/Python/Node.js/...

Part 1: Download and Install Orient DB

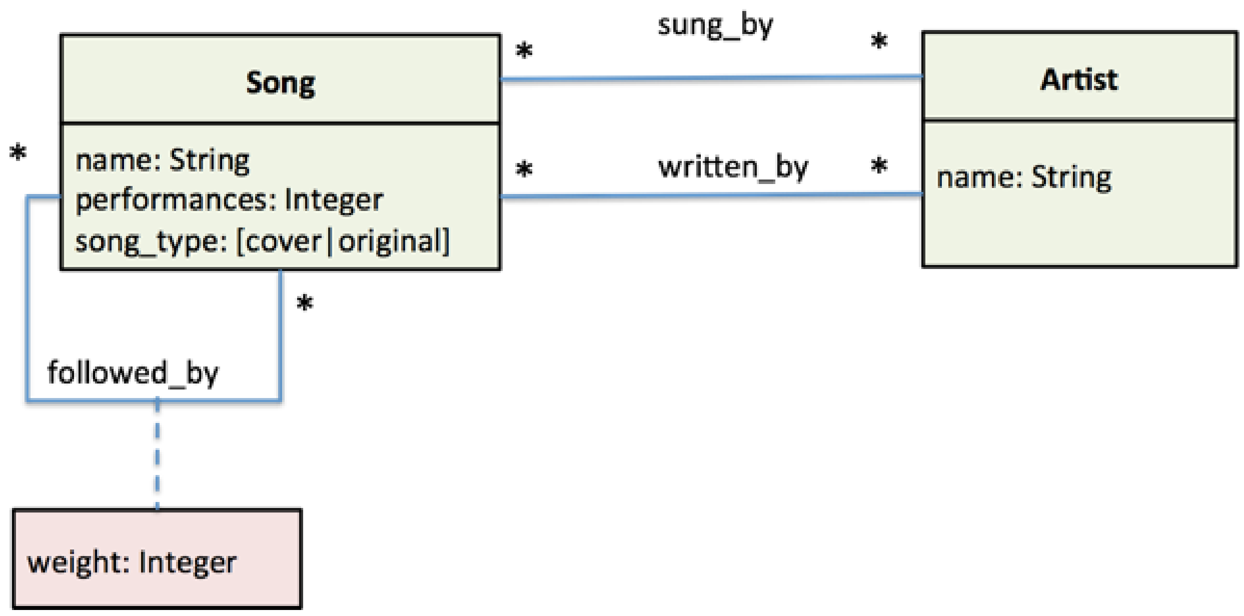
I've prepared a simple video that explains how to download and install Orient DB. I'm on a Mac, hence the video shows how to install it on Macs. I do, however, think it should be easy transpose the steps to Windows or Linux.

At the time I created this video, the latest version of Orient DB was 1.6.2 (things are moving rapidly at OrientDB, and I noticed they already released 1.6.3 before I got a chance to publish the blog). Orient DB releases many versions per year, so there is a risk that at the time you read this tutorial, my instructions may be slightly out of date. All the steps that I followed in the video can be found on the Orient Technologies website [http://www.orientechnologies.com/] and I'm sure it will be kept up to date. Hence, if the version you downloaded is different from 1.6.2, you may be better off following the up-to-date instructions on the Orient DB sites.

A bit of warning: Towards the end I do execute a few queries to ensure that everything works. Don't despair if you don't understand what I'm doing. All I wanted to was ensure that I'd succeeded in installing Orient DB.

Part 2: Play Around with the Grateful Dead Graph

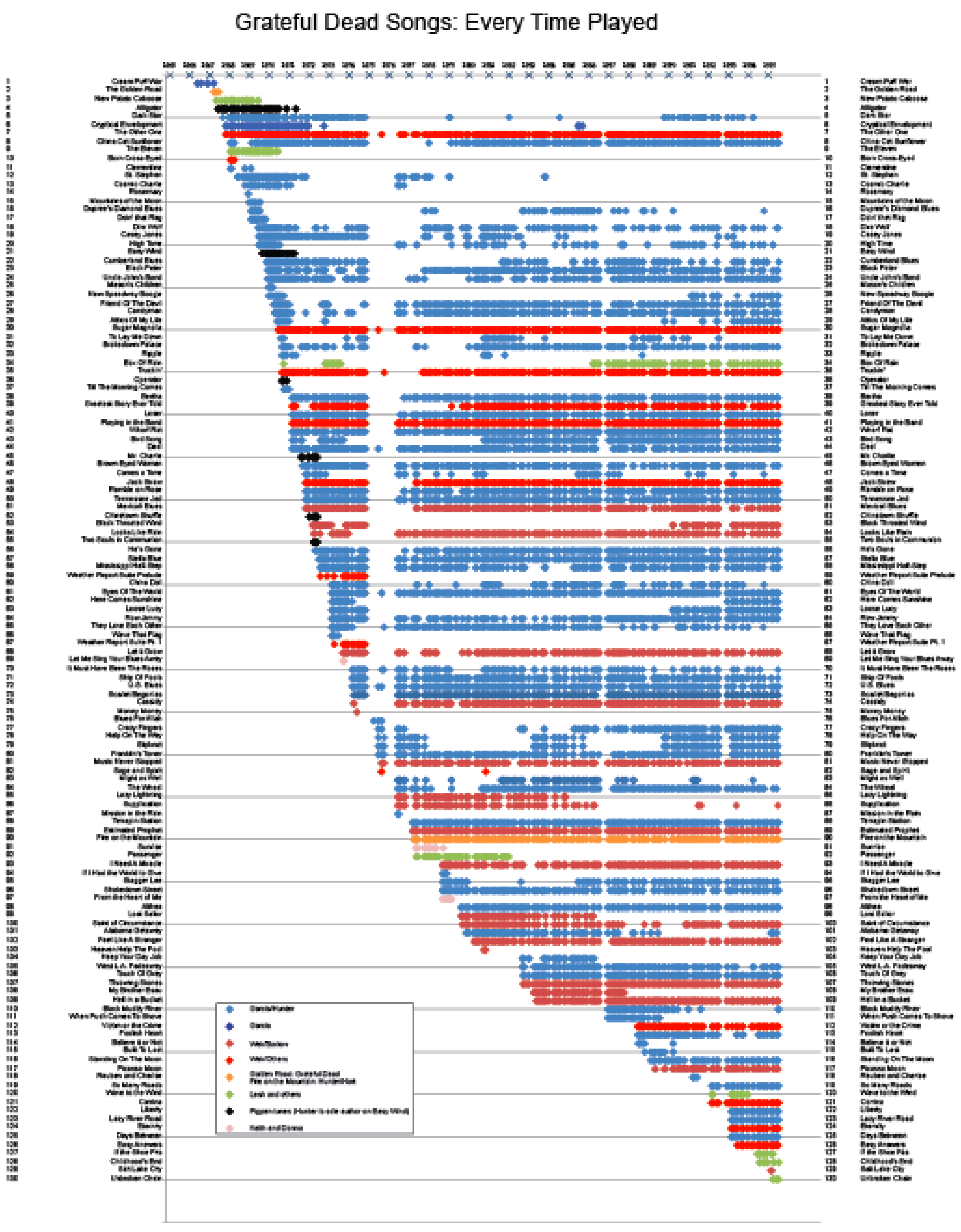
Orient DB comes with an example database instance. The provided database contains a graph defining performances of Grateful Dead. The graph is sparse and the information model is quite simple. This is unfortunate as it is hard to illustrate all the power of Orient DB with this graph. Perhaps at a later time I'll try to find an open source dataset that has more interesting information (suggestions welcomed!).
The Grateful Dead graph can be illustrated with the following UML model:



The model above is as truthful as I could make it. I did take exercise some artistic 'enhancements'. This to make the model easier to read. Here are my embellishments:

- Notice the attribute on the ‘followed_by’ association. Vrious versions of UML have provided different ways to depicting attributes on relationships. Let me explain what I mean by my way of using UML. The example database stores a count the number of times a song has been followed by another song in a concert (I did not create the model, so I’m providing my own interpretation). The weight is a count of the number of times. To illustrate this in UML I have to show properties on associations. Conceptually, this is not a problem for a Graph Database because it can store properties on edges. It may, however, awkward to see the mapping to other technologies.
- I decided to expand the enumerated type specifying the "song_type" . A more common way to model this would be the use of an enum-type. However, I think ‘display-wise’ the expanding the enum is better here as it is only used in one place.
- Another ‘cheat’ is the introduction of classes. The example database do not actually use custom classes (all the objects are instances of a class called ‘V’; short for vertex). They do, however, provide a property called ‘type’ that is used for the purpose of typing. I believe the model I created is better at illustrating the original intent of the model.

I would assume only a percentage of the readers are familiar with Grateful Dead [http://en.wikipedia.org/wiki/Grateful_Dead] , so let me give you a few sentences about them. Grateful Dead is a cult rock band that started playing in California in the 60’s. I assume the graph is populated from the same data set that produced this illustration:



It is really not important who the Grateful Dead [http://en.wikipedia.org/wiki/Grateful_Dead] were and whether you like them or not. Just think of them as some rock group that performed a set of songs very many times. Someone cared enough to capture when what songs they performed, what song followed another song, who wrote the song, and who sung the song.

With that introduction, let’s move on to the video where I create a set of queries and navigate around the example database.

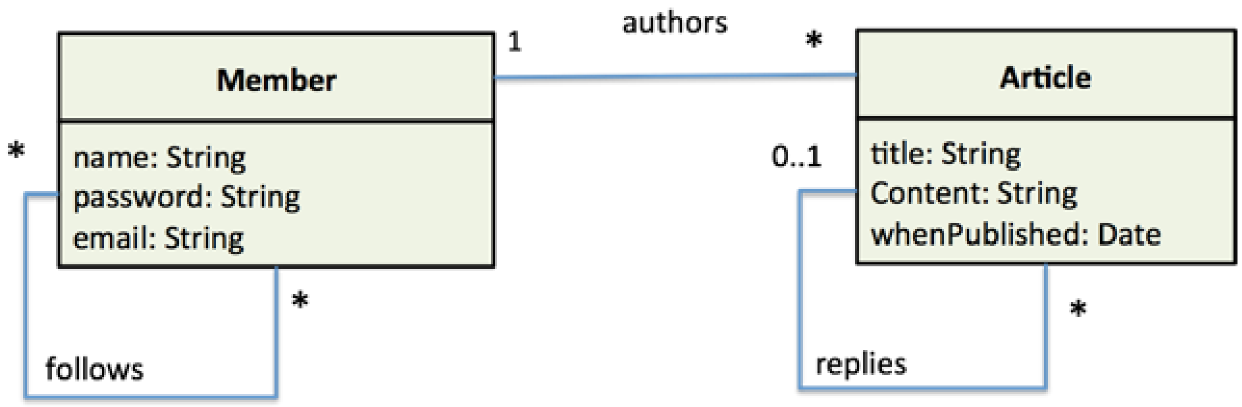
Just for your reference, here are some of the queries I used

- Select all vertices (or object) in the database
  - `select * from V`
- Select the vertex with the id #9:8
  - `select * from V where @rid=#9:8`
  - This could also be written as:
    - `select * from #9:8`
- Select all the artists
  - `select * from V where type=‘artist'`
- Select all the songs that have been performed more than 10 times
  - `select * from V where type = 'song’ and performances > 10`
- Count all songs
  - `select count(*) from V where type='song'`
- Count all artists
  - `select count(*) from V where type=‘artist'`
- Find all songs sung by the artist with the id #9:8 (notice, the result will include the artist with id #9:8)
  - `traverse in_sung_by from (select * from V where @rid=#9:8)`
- Find only songs sung by the artist with id #9:8 (only songs)
  - `select * from (traverse in_sung_by from (select * from V where @rid=#9:8)) where type=‘song’`

- This could also have been written as:
  - `select expand(set(in('sung_by'))) from #9:8`
- Find authors of songs sung by the artist with the id #9:8
  - `select * from ( traverse out_written_by from (select * from ( traverse in_sung_by from ( select * from V where @rid=#9:8 ) ) where type='song') ) where type=‘artist'`
  - Or more effectively:
    - `select expand(set(in('sung_by').out('written_by'))) from #9:8`

Creating Our Own Database

In this section, I'll explore how you can create your own database. I'll be using custom classes in this tutorial (unlike the Grateful Dead database that only used the built in 'V' class).



The following video shows how we can create a database that uses the above model as its schema:

Below, I've included the statements required to create the database used in the above video.

```
create class Member extends V

create property Member.name string

alter property Member.name min 3

create property Member.password string

create  property Member.email string

create class Article extends V

create property Article.title string

alter property Article.title min 3

create property Article.title string

create class follows extends E

create class replies extends E

create class authors extends E
```

Populating the database

In the previous section we defined a database schema. That means that we've prepared Orient DB to store and enforce constraints on objects we'll insert into this database. It would be possible to store the same information without the schema (in so called schema-free mode). However, since we decided to go down the route of schema-full use, Orient DB can help us enforce the constraints on the data structures.

The video below explores different ways of inserting data into the schema we created in the previous step.

If you don't want to sit through the video, I've included some functions/key script fragments below the video for you to instantiate the database as you please.

Important: For some of the functions that I show in the video to work, you have to add a handler to the Orient DB-server-config.xml. Please add the following XML fragment to the oriented-server-config.xml file (this step is covered in the video):

```
<handler class="com.orientechnologies.orient.graph.handler.OGraphServerHandler">
  <parameters />
</handler>
```

Why are you doing the above step you may wonder? It is to ensure that the default handler used in the JavaScript functions is of the graph type.

Some key syntax for inserting data:

```
    create vertex {Vertex Class Name} set ({propertyname=value},…)

    create edge {EDGE CLASS NAME} from {OUT_VERTEX_ID} to {IN VERTEX ID}
```

Part 5: Querying the Database (Again)

Now that we have some data (assuming you finished step 4), let’s see if we can define some interesting queries. If you want to try it out without seeing what I do, here are some suggestions for queries to formulate:

- Given a member X:
  - Lookup everyone that follows X.
  - Lookup all articles that X posted that was eventually replied to.
  - Lookup everyone that at one time participated in an article that originated with a member X.
- Find all members that have answered some article posted by someone else.
- If I changed the content of an article, who would be effected by this (the members that posted articles prior to this article and the members posting articles as relies to this article.

Summary

In this post I've tried to give you a head start on Orient DB. Most of my use of Orient DB in this tutorial has been graph-oriented. If time permits, I'll try to make another tutorial later that expose the power of Orient DB as a document database.

I hope you learned something. Please don't feel shy about leaving comments. I actually moderate the comments as I've had a very large number of spam posts. If you're comment doesn't appear immediately, please do not disappear. As soon as I get a chance to make sure the comment is not spam, I'll allow it and try to answer you.

Posted 5th January 2014 by Petter Graff

9 View comments

dan 3:27 PM

Petter very nice job, I have been working with OrientDb for a little while now. I made a little custom rest adapter to allow for using isomorphics smartclient framework with Orient. One of my biggest problems now in actually building a database is schema design coming from a relational and going to graph. Not alot of docs beyond simple examples. I am looking at designing a CRM app with Orient and still struggling with graph schema,design like with graph when does link and linkset come into play vs just an edge, but you tutorial has helped. Would love to see more on this subject, Nice job!

Thanks,
Dan

Reply

Ven Karri 6:53 PM

Excellent tutorial. thanks for posting.

Reply

Quang Kin 7:29 AM

Hi,
I need for help!
Assume I have query : select Mail, expand(both('Friend')) from User where Name='hoang'
Query return all field. I only want to get field Name, Mail. How can i do that.
Thank,

Reply

Replies

Petter Graff 11:28 AM

Can't you just wrap a simple query around your other query?

Reply

Greymantle 12:54 PM

Really excellent tutorial, thanks for taking the time to put this together! I was able to follow along perfectly until 8:14 of Tutorial video #4, when I got the following error:

"Error on parsing script at position #0: Error on execution of the script Script: createSomeMembers ------^ sun.org.mozilla.javascript.internal.EcmaError: ReferenceError: "gdb" is not defined. (#11) in at line number 11"

It seems obvious that the reference to the database is broken, but I cannot find anywhere in the OrientDB docs where a "gdb.save()" command exists, and the complexity of SQL vs graph is a bit confusing when searching in the docs for the right spot.

But in spite of this current roadblock, your tutorial has been really excellent and is very much appreciated! Thank you!

Reply

Replies

Petter Graff 2:25 PM

Did you setup the graphical database? I show you how to do that in the first video...

Sudara Narangoda 6:11 AM

same error occured when executing the function..

Reply

Petter Graff 2:24 PM

This comment has been removed by the author.

Reply

john gonzalez 8:14 AM

https://groups.google.com/forum/#!topic/orient-database/BOueQD8hMOA

Any thought on this ?

Reply

Enter your comment...

Comment as: Arunkumar Kri ▼ Sign out

Publish Preview ☐ Notify me