Big Data

NoSQL Database Types: episode II

ORDINA

CONNECTIVATE

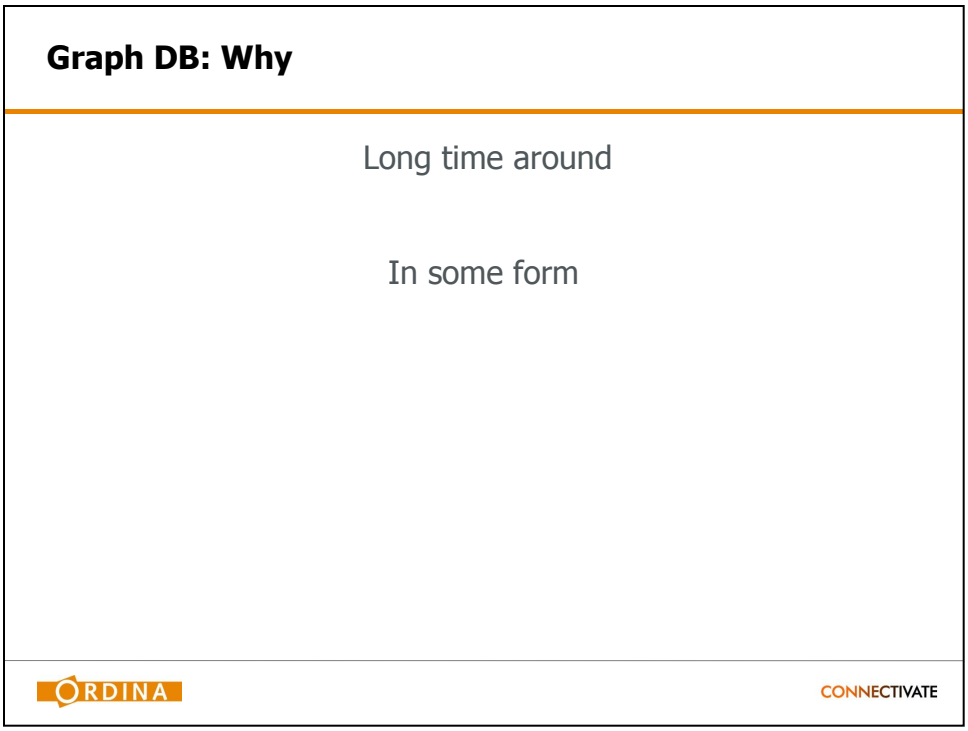
Content

- Document Store
- Graph DB

Graph

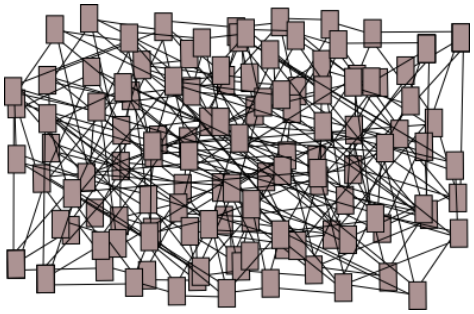
Graph DB

- Why Graph DB
- OrientDB
- OrientDB vs Neo4J



First databases were navigational databases … where records were found by following references from other objects, a lot like graph databases.

Graph DB: Why



ORDINA CONNECTIVATE

One of the main reasons why RDBMS users resist to go over is because of the complexity of the RDBMS model.

Graph DB: Why

Can it handle complexity?

- Key/Value
- Column Store
- Document Store

} can not handle relations

- Graph Database !

ORDINA CONNECTIVATE

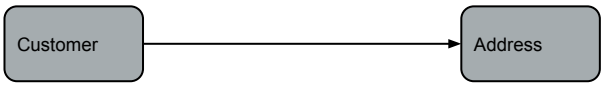
No SQL great for big data and data with flexible formats but …

Graph DB: Why

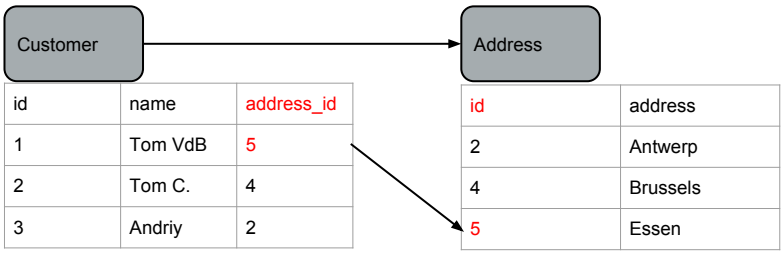
ORDINA CONNECTIVATE

To understand why lots of nosql do not support relations we will look how an RDBMS manages it

Graph DB: RDBMS relations



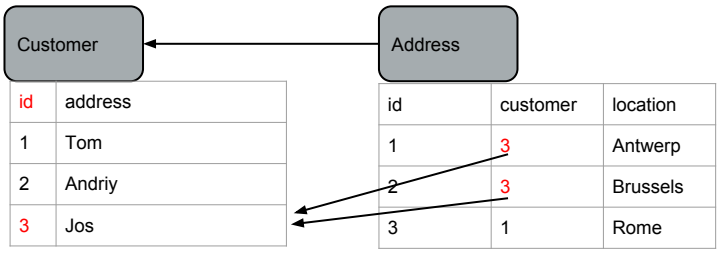
Graph DB: 1 to 1



id	name	address_id
1	Tom VdB	5
2	Tom C.	4
3	Andriy	2

id	address
2	Antwerp
4	Brussels
5	Essen

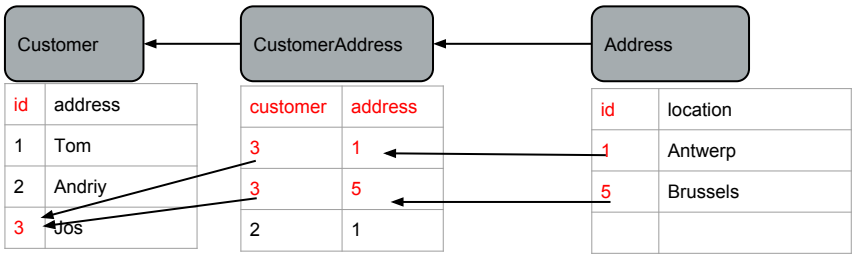
Graph DB: 1 to N



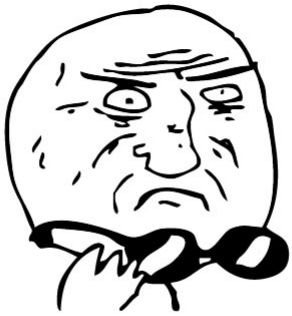
id	address
1	Tom
2	Andriy
3	Jos

id	customer	location
1	3	Antwerp
2	3	Brussels
3	1	Rome

Graph DB: N to M



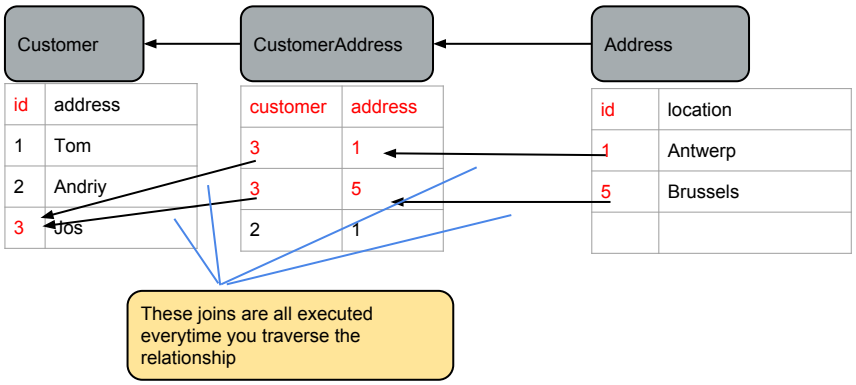
Graph DB: what is wrong



ORDINA CONNECTIVATE

What is wrong with the relational model?

Graph DB: The join



Graph DB: what is wrong

ORDINA CONNECTIVATE

What is wrong with that

Graph DB: what is wrong

ORDINA CONNECTIVATE

A join means searching for a key in another table.

Graph DB: what is wrong

ORDINA CONNECTIVATE

In order to improve performance one adds indexing.

Graph DB: what is wrong

ORDINA CONNECTIVATE

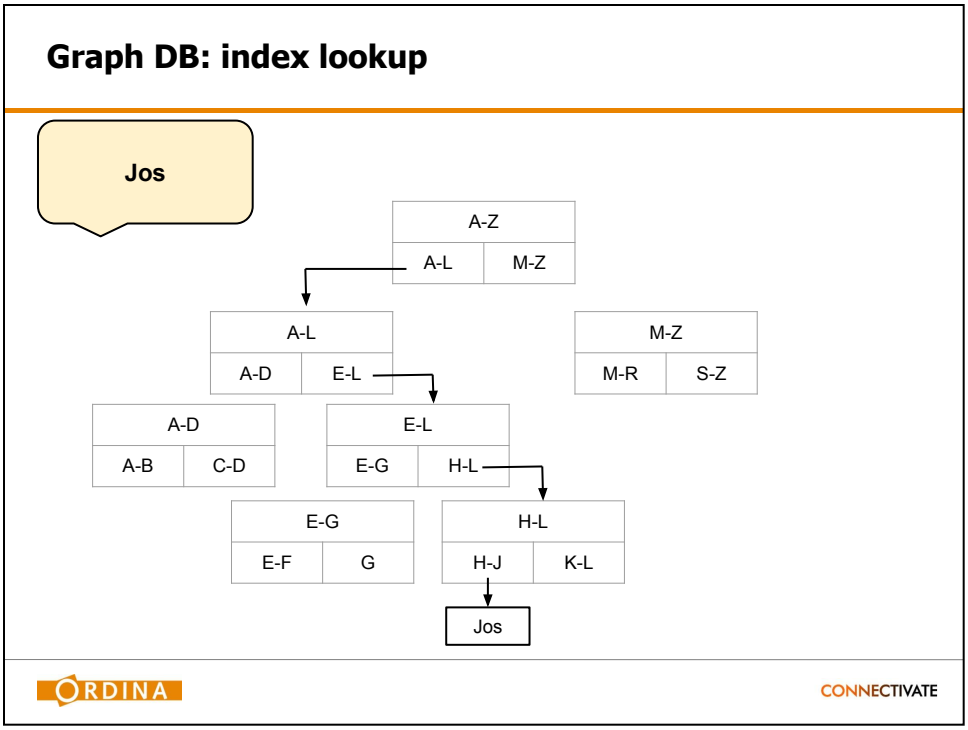
But indexing might improve read performance, it will slow down insert, updates and deletes

Graph DB: what is wrong

A join is essentially a lookup into an index.

This is done for every single join.

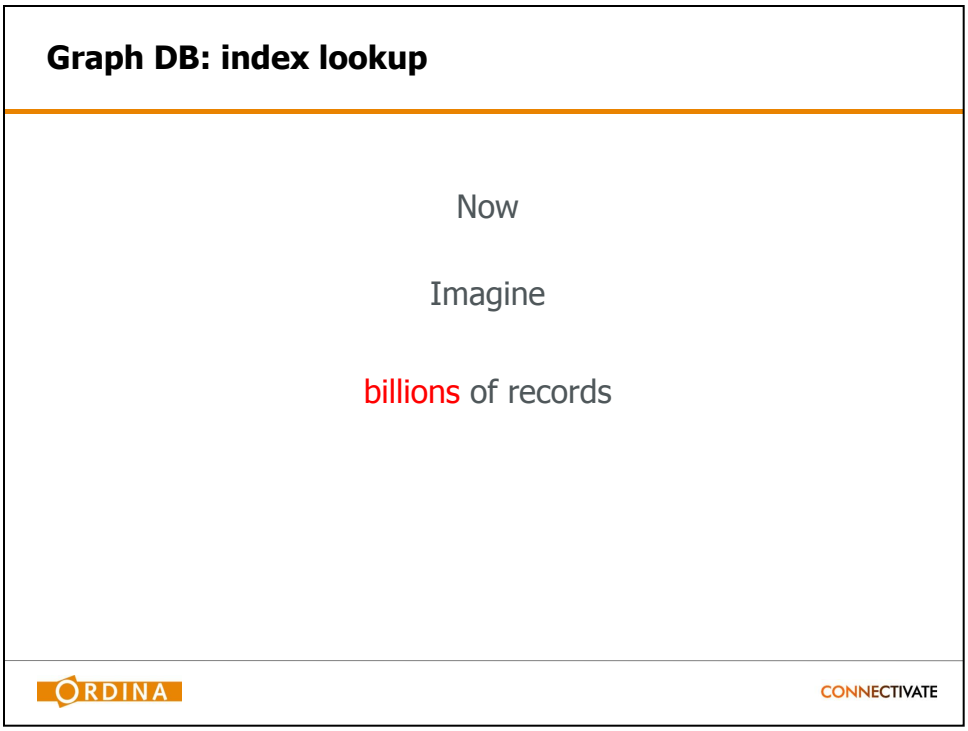
If you are traversing hundreds of relations you are executing hundreds of joins.



Index lookup: how does it work

- Let's go and find Jos
- This is our starting point, let's go down the tree

Found after 5 steps: with millions of indexed record the tree depth could be 1000s of levels



Now imagine how many steps a lookup operation would take for an index with billions of records.

Graph DB: index lookup

This join is executed
for every involved table
multiplied
for all scanned records

ORDINA CONNECTIVATE

which is why the performance of your database suffers when it becomes bigger and bigger and bigger …

Graph DB: What about document databases

```
{
 "_id": 1,
 "name": Tom,
 "address_id": 4
}
```

ORDINA CONNECTIVATE

similar approach is used as with RDBMS => id for the relation is stored and looked up

work around is to store that information within the document, but that is not always possible and requires you to pay attention on data correctness

Graph DB: Is there a better way

"A graph database is any storage system that provides index-free adjacency "

Marko Rodriguez

"auther of Tinkerpop Blueprints"

ORDINA CONNECTIVATE

This means that every element contains a direct pointer to its adjacent elements and no index lookups are necessary.

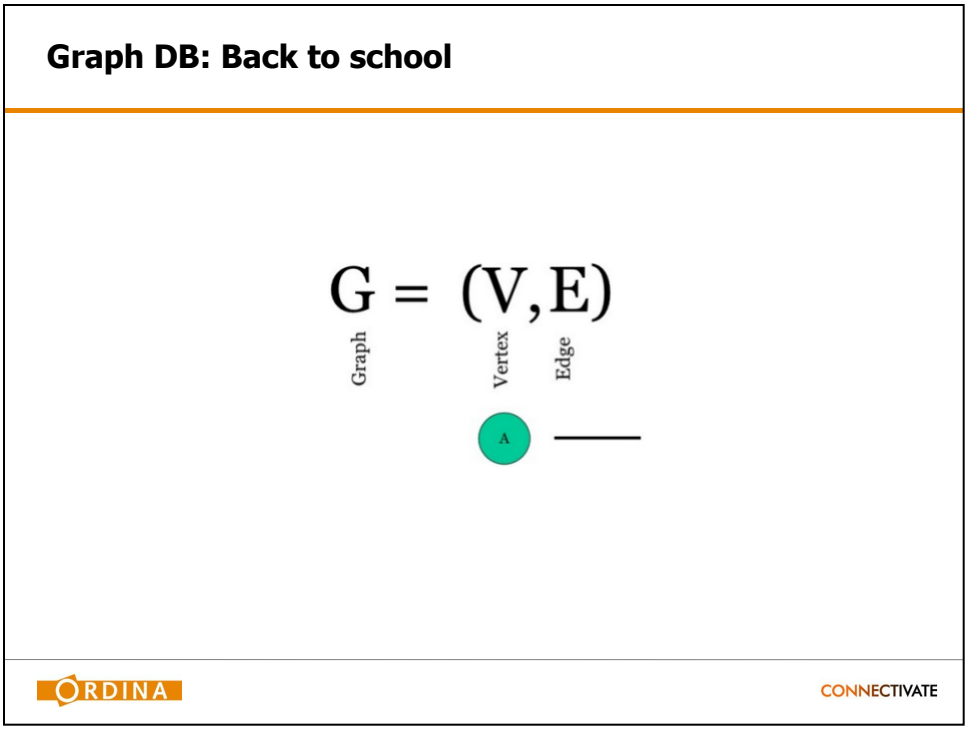
Graph DB: Is there a better way

Index free relationshops ?

ORDINA CONNECTIVATE

Every developer knows the relational model - but what about the Graph one?

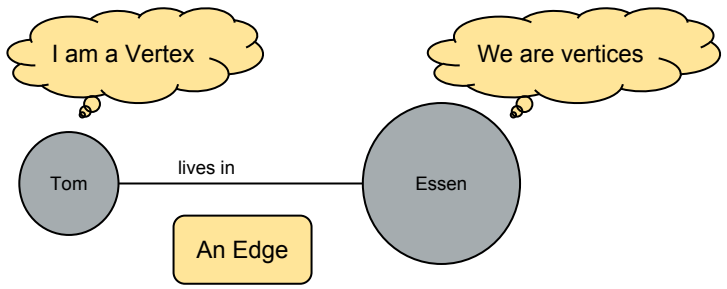
How does a graph DB manage index free relationshops ?



Graph theory crash course

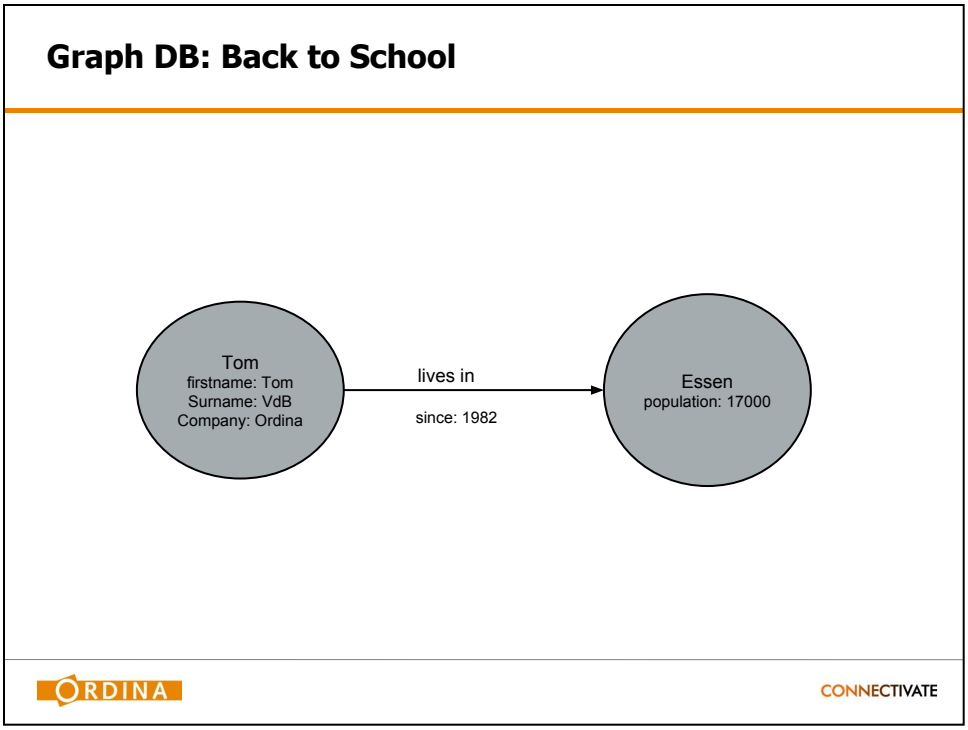
- **Graph**: An object that contains vertices and edges.
    - **Element**: An object that can have any number of key/value pairs associated with it (i.e. properties)
        - **Vertex**: An object that has incoming and outgoing edges.
        - **Edge**: An object that has a tail and head vertex.

Graph DB: Back to School



ORDINA CONNECTIVATE

Basic graph



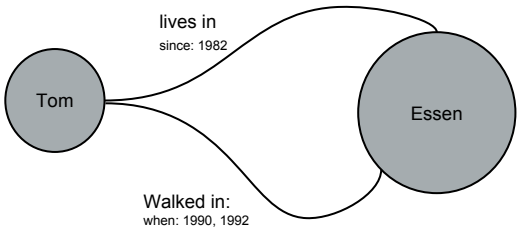
The propety Graph Model

Vertexes can have properties
Edges are directed

* https://github.com/tinkerpop/blueprints/wiki/Property-Graph-Model

Graph DB: Back to School

1 to N relationships



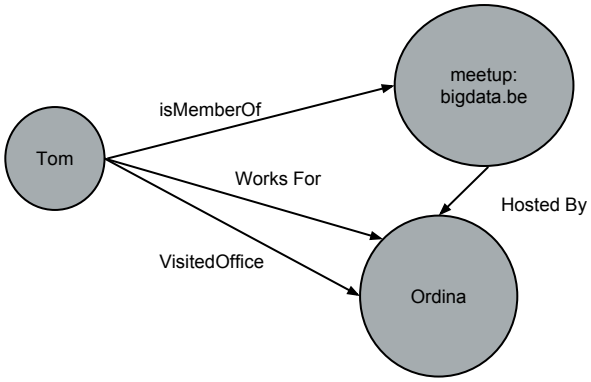
ORDINA CONNECTIVATE

An edge only connects 2 vertices

Use multiple edges to represent 1-N and N-M relations

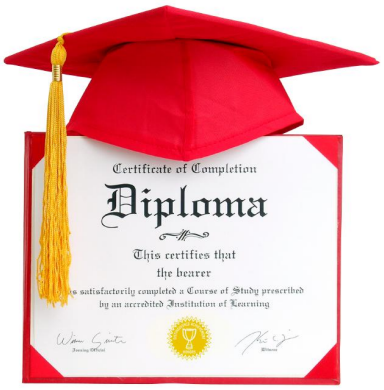
Graph DB: Back to School

Graph Example



Graph DB: Back to school

Congratulations - you are now graduated in graph theory

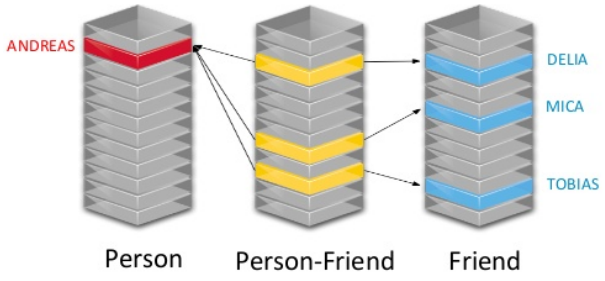


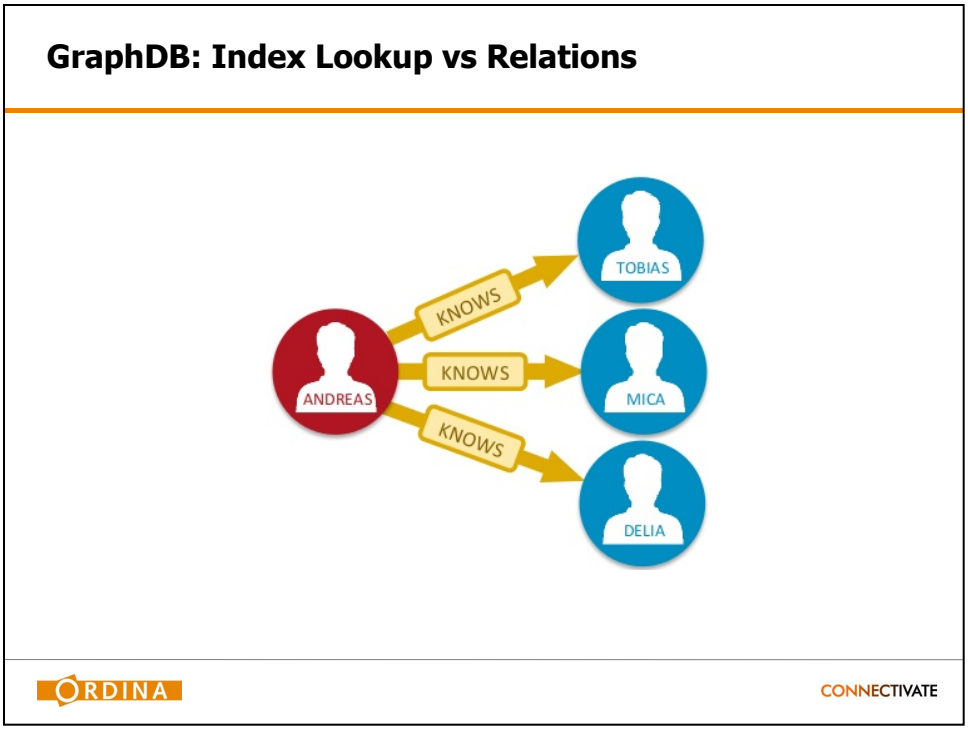
ORDINA

CONNECTIVATE

The graph theory is so simple yet so powerfull

GraphDB: Index Lookup vs Relations





A graphdb handles relations as a physical link to the record while a relational databases computes the relationship every time you do a query

(computes => via binary tree / otherwhise it would be a full table scan)

Graph DB: OrientDB

- How does OrientDB manage relationships
- Some Limits
- Hybrid
- Transactions and ACID
- Create the Graph
- Query vs Traversal
- Schema

OrientDB: Manage Relationships

ORDINA

CONNECTIVATE

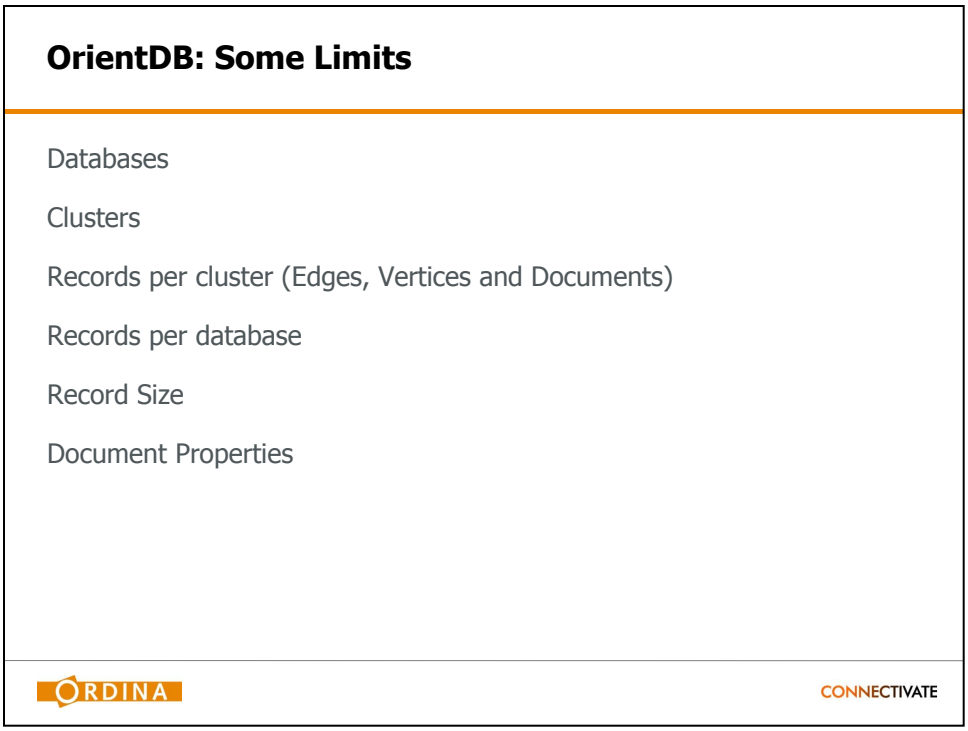
Rid - record id is the physical position of the Vertex

OrientDB: Manage Relationships

ORDINA CONNECTIVATE

The record id of the edge is saved in both vertices as in or out

So it can be followed from both vertices
*via the outgoing indication
* or via the incoming one

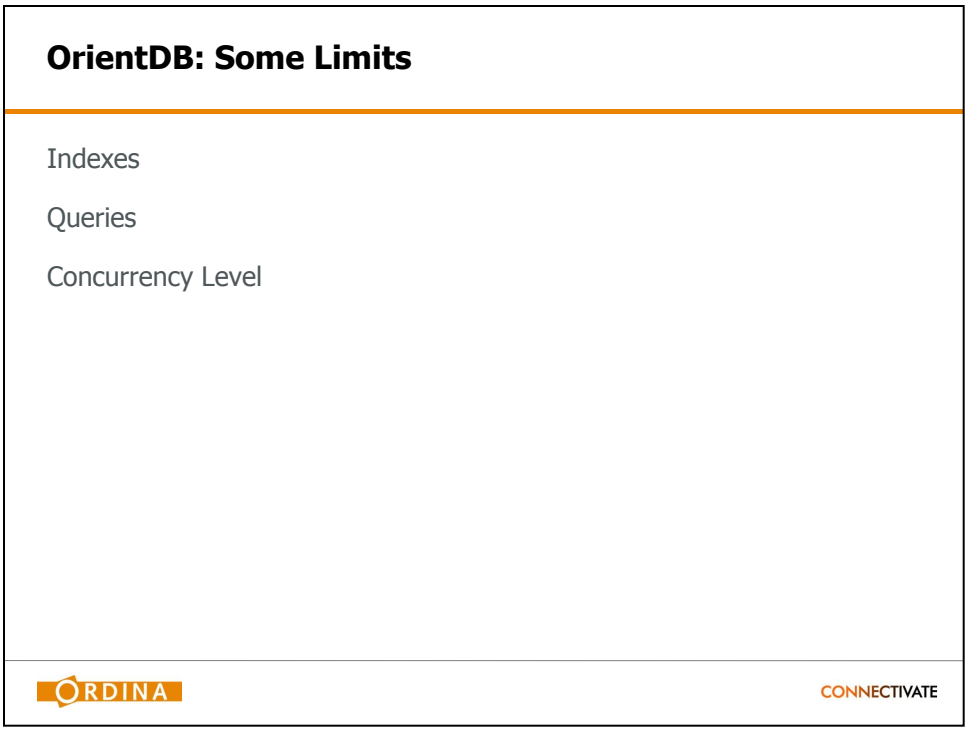


- There is no limit to the number of databases per server or embedded. Users reported no problem with 1000 databases open
- each database can have a maximum of 32,767 clusters (2^15-1)
- Records per cluster (**Documents**, **Vertices** and **Edges** are stored as records): can be up to 9,223,372,036,854,780,000 (2^63-1), namely 9,223,372 Trillion records

A Cluster: a place where records are stored, can best be compared to a table in a relational table

- **Records** per database (**Documents**, **Vertices** and **Edges** are stored as records): can be up to 302,231,454,903,000,000,000,000 (2^78-1), namely 302,231,454,903 Trillion records
- **Record size**: up to 2GB each, even if we suggest avoiding the creation of records larger than 10MB. They can be split into smaller records
- **Document Properties** can be:
  - up to 2 Billion per database for schema-full properties
  - there is no limitation regarding the number of properties in schema-less mode. The only concrete limit is the size of the

- Document where they can be stored. Users have reported no problems working with documents made of 15,000 properties

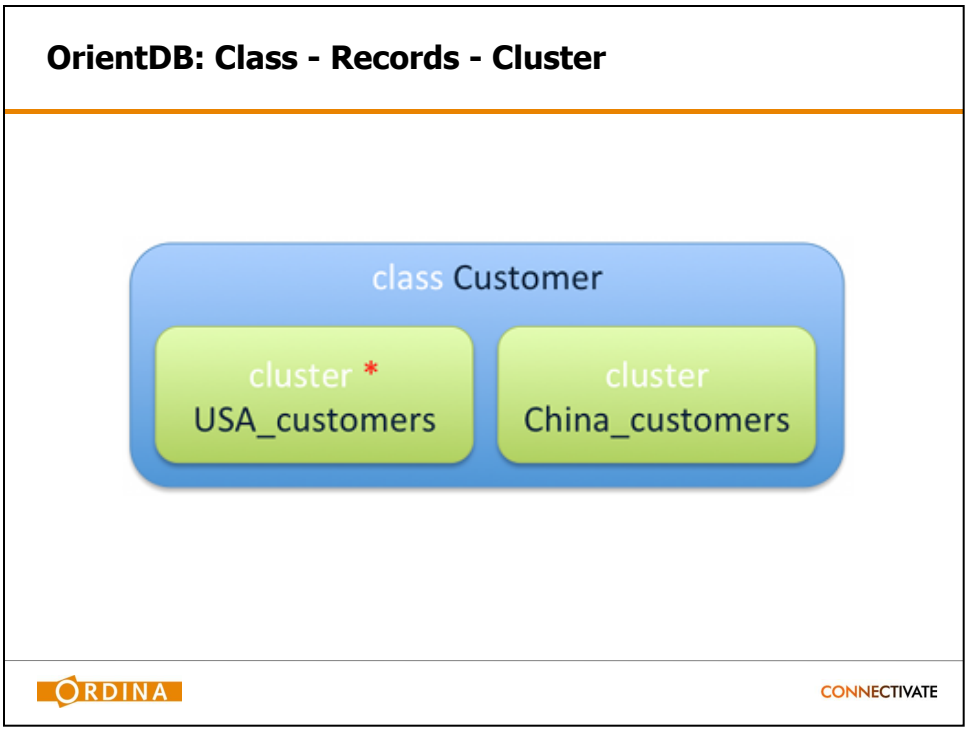


- **Indexes** can be up to 2 Billion per database. There are no limitations regarding the number of indexes per class
- **Queries** can return a maximum of 2 Billion rows, no matter the number of the properties per record
- **Concurrency level**: in order to guarantee atomicity and consistency, OrientDB acquire an exclusive lock on the storage during transaction commit. This means transactions are serialized. Giving this limitation, *the OrientDB team is already working on improving parallelism to achieve better scalability on multi-core machines by optimizing internal structure to avoid exclusive locking.*

OrientDB supports optimistic transactions, so no lock is kept when a transaction is running, but at commit time each record (document, or graph element) version is checked to see if there has been an update by another

client.

-
-
-



Vertices and Edges are stored as records => with a class type. Generally seen a class is linked to a single cluster (a place where records are stored) => but this can also be more. (a default one can be specified)

The benefits of using different physical places to store records are:

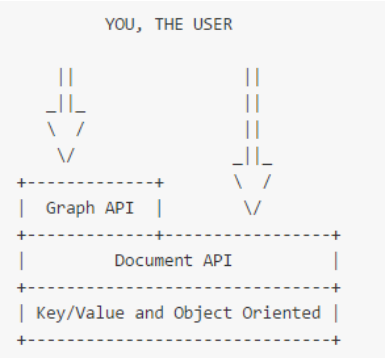
- faster queries against clusters because only a sub-set of all the class's clusters must be searched
- good partitioning allows you to reduce/remove the use of indexes
- parallel queries if on multiple disks
- sharding large data sets across multiple disks or server instances

There are two types of clusters:

- **Physical Cluster** (known as **local**) which is persistent because it writes directly to the file system
- **Memory Cluster** where everything is volatile and will be lost on

- termination of the process or server if the database is remote

OrientDB: Hybrid Model



ORDINA CONNECTIVATE

Some of you might have noticed that I mentioned Documents

In OrientDB the graph model is built on top of the document model

- The Document API is simpler than the Graph API in general.
- Relationships are only Mono Directional. If you need Bidirectional relationships, it is your responsibility to maintain both LINKs.
- A Document is an atomic unit, while with Graphs everything is connected as In & Out. For this reason, Graph operations must be done within Transactions. Instead, when you create a relationship between documents with a LINK, the target linked document is not involved in this operation. This results in better Multi-Thread support, especially with insert, deletes and updates operations.

The document model also requires less objects => less traverse time

But since this is a talk about graphs we are not going deeper into that

OrientDB: Transactions and ACID

ORDINA CONNECTIVATE

By default no transactions.

Transactions are client-side until commit

OrientDB: Transactions and ACID

ORDINA CONNECTIVATE

If a records gets updated with stale data => error will be returned

OrientDB: Transactions and ACID

ORDINA CONNECTIVATE

Distributed transactions: use of quoram (majority) untill commit => if quorum not reached, rollback.

Before rollback it is possible that updated records are returned by the system.

OrientDB: Create the Graph - SQL

```
orientdb> create vertex Customer set name = 'Tom'
Created vertex #13:35

orientdb> create vertex Address set name = 'Essen'
Created vertex #13:100

orientdb> create edge Lives from #13:35 to #13:100
Created edge #14:54

orientdb> select in('Lives') from Address where name = 'Essen'
```

ORDINA CONNECTIVATE

OrientDB uses SQL as it’s domain specific language DSL

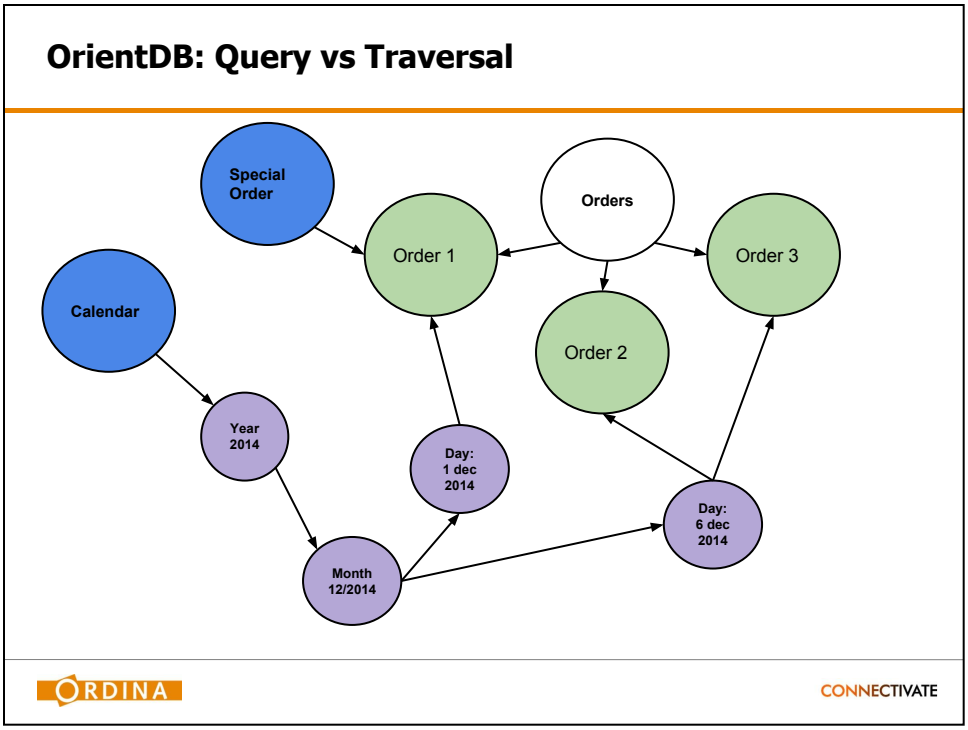
The in are the incoming vertices

OrientDB: Create the Graph - Java

```
1  Graph graph = new OrientGraph("local:/tmp/db/graph");
2
3  Vertex tom = graph.addVertex("class: Customer");
4  tom.setProperty("name", "Tom");
5
6  Vertex essen = graph.addVertex("class: Address");
7  essen.setProperty("name", "Essen");
8
9  Edge edge = graph.addEdge("Lives in", essen);
```

ORDINA CONNECTIVATE

In Java it looks like this



With a well connected database in the form of a super graph you can cross records in stead of traversing them.

***All you need are a few Root Vertices**

***Root vertices can be enriched with Meta Graphs to decorate graphs with additional information and make retrieval easier/faster**

OrientDB: Schema

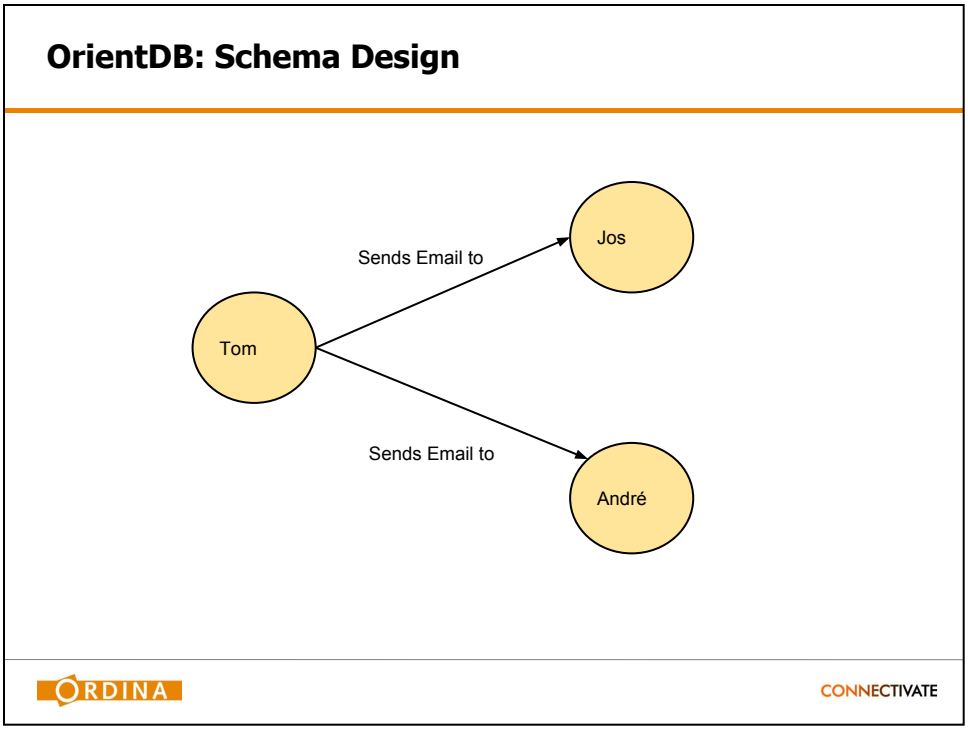
- schema full
- schema-mixed
- schema-less

ORDINA CONNECTIVATE

Schema with constraints on fields an validation rules

Schema mixed: with mandatory and optional fields + constraints: best of both worlds

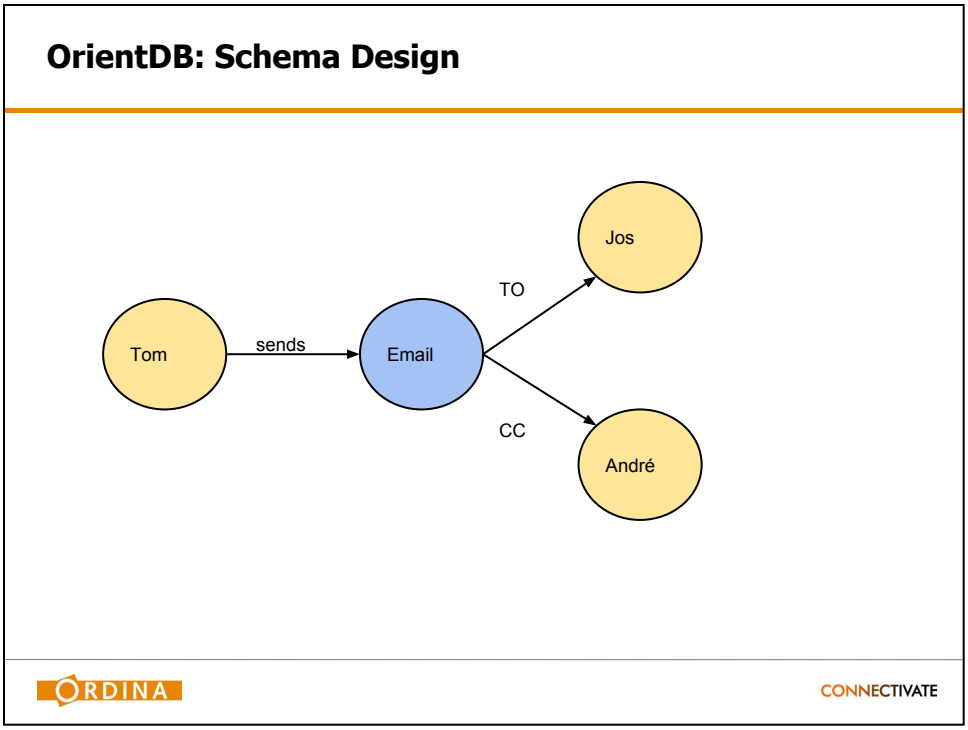
Schema Less:



Thinking about your schema is important

For example: Sending an email

Simple schema - but you might lose content, what of the body/subject of the mail, were other persons also in that email?



Depending on your use case - this might be better

Thinking about your schema is, like with other nosql databases, very important.

Think about the use case and not just - normalize

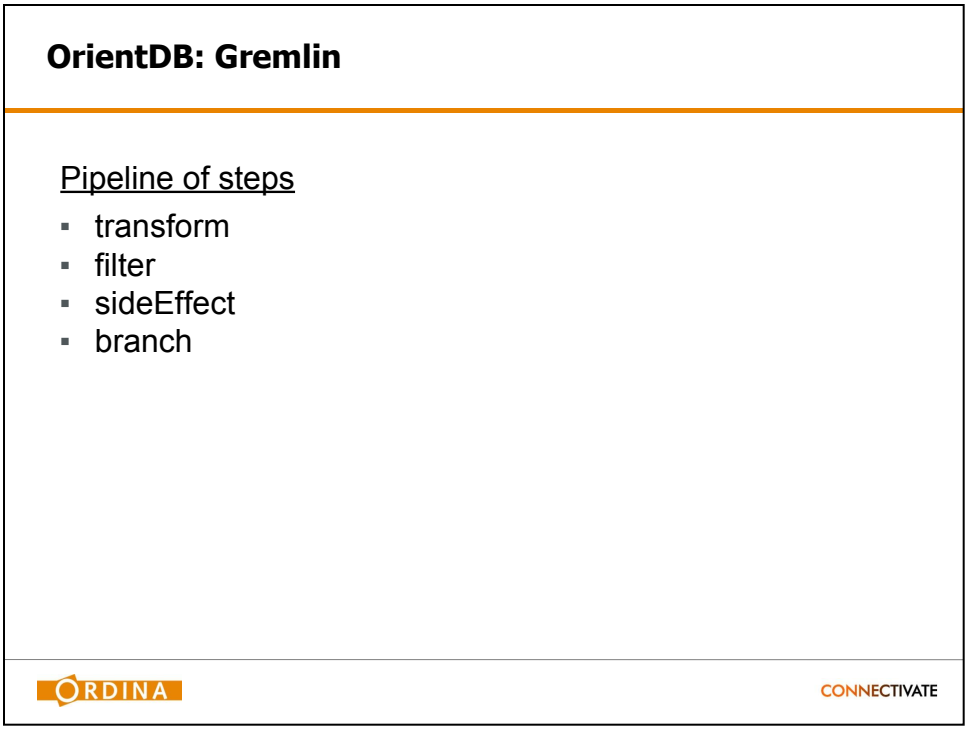
OrientDB: Gremlin

ORDINA CONNECTIVATE

Gremlin is a graph traversal scripting language.

Native java / groovy support

Works on graph databases who have implemented the blueprints property graph data model



- `step`: a generic, general-purpose computational step.
  - `transform`: take an object and emit a transformation of it.
  - `filter`: decide whether to allow an object to pass or not.
  - `sideEffect`: pass the object, but yield some side effect.
  - `branch`: decide which step to take.

OrientDB: Gremlin

Query	SQL	Gremlin
Select by equality	`select * from users where age = 35`	`g.V('type', 'user').has('age', 35).map()`
Select by comparison	`select * from users where age > 21`	`g.V('type', 'user').has('age', T.gt, 21).map()`
Select by multiple criteria	`select * from users where sex = "M" and age > 25`	`g.V('type', 'user').has('age', T.gt, 25).has('sex', 'M').map()`
Order by age *(switch 'a' and 'b' to do asc)*	`select * from users order by age desc`	`g.V('type', 'user').order({ it.b.getProperty('age') <=> it.a.getProperty('age') }).map()`
Paging	`select * from users order by age desc limit 5 offset 5`	`g.V('type', 'user').order({ it.b.getProperty('age') <=> it.a.getProperty('age') })[5..<10].map()`

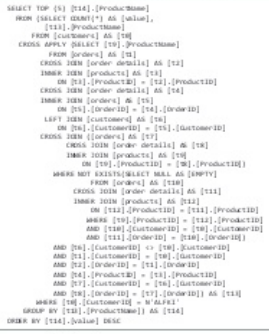
ORDINA CONNECTIVATE

Some examples

OrientDB: Gremlin

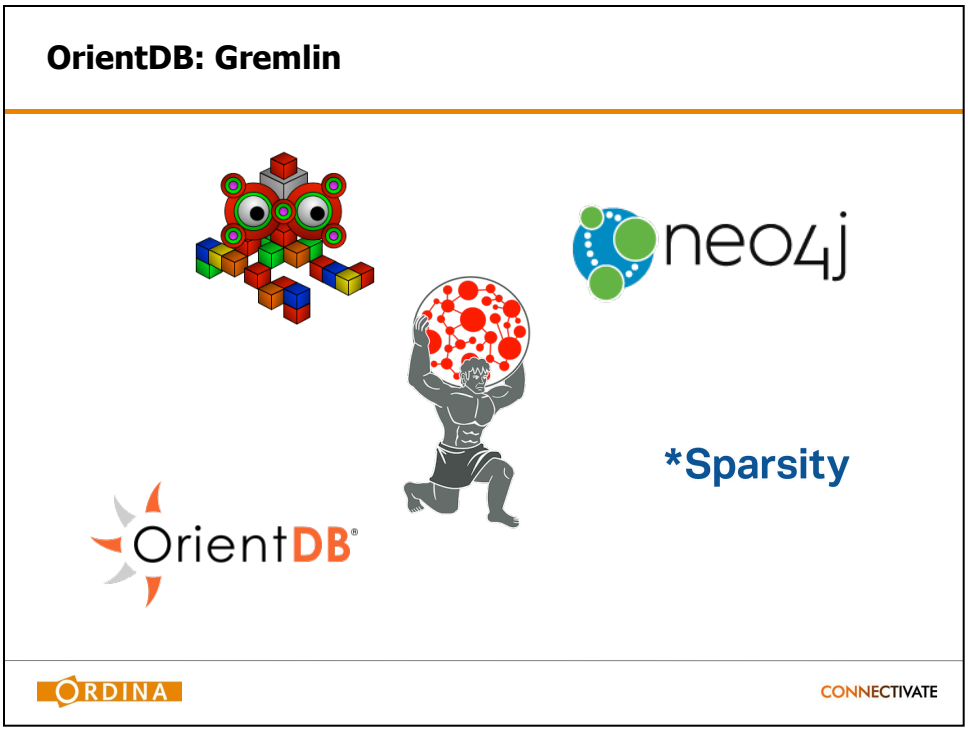
Query	SQL	Gremlin

Join	`select users.* from users`
	`  inner join groups`
	`  on users.gId = groups.id`
	`  where groups.name = "devs"`
		`g.V('type', 'groups')`
		`  .has('name', 'dev')`
		`  .in('inGroup').map()`

Join-on-join-on-join …		`g.V('customerId','ALFKI')`
		`  .as('customer')`
		`  .out('ordered')`
		`  .out('contains')`
		`  .out('is')`
		`  .as('products')`
		`  .in('is')`
		`  .in('contains')`
		`  .in('ordered')`
		`  .except('customer')`
		`  .out('ordered')`
		`  .out('contains')`
		`  .out('is')`
		`  .except('products')`
		`.groupCount().cap()`
		`.orderMap(T.decr[0..<5]`
		`.productName`

ORDINA CONNECTIVATE

We will not go deeper into gremlin but it is nice to let you know it exists and can be used on a lot of graph db, for example …



Tinkergraph

Neo4J - n1 graph database

Titan - n3, but company who was doing most of the development was taken over bij datastax (company behind cassandra)

OrientDB - n2

Sparksee - graph db from sparsity

Graphdb: Use Cases

- Recommendation engines
- Ranking/Credibility
- Path Finding
  (shortest, longest, mutual friends)
- Social
  (friendship, following, key connectors)

amazon.com

Google

LinkedIn

facebook

ORDINA

CONNECTIVATE

Please note: not all recommendation systems are “real” => lots of retailers allow companies to choose the “people looking for this product also looked for … “ (more expensive comparable products) and “people buying this product also bought ... “ (accessories)

Some code to play with

1. Go to https://github.com/tomvdbulck/orientdb_initiation
2. Make sure the following items have been installed on your machine:
   - Java 7 or higher
   - Git *(if you like a pretty interface to deal with git, try SourceTree)*
   - Maven
3. Install VirtualBox https://www.virtualbox.org/wiki/Downloads
4. Install Vagrant https://www.vagrantup.com/downloads.html
5. Clone the repository into your workspace
6. Open a command prompt, go to the vagrant folder and run

   ```
   vagrant up
   ```
7. This will start up the vagrant box. The first time will take a while (approx. 5 min) as it has to download the OS image and other dependencies.

Want More?

ORDINA CONNECTIVATE

yes, we are currently churning out lots of events at the big data competence center with 2 workshops and 2 meetups within 1 month.

Even More?

Upcoming **meetup** on170/06 - @ Ordina

1st meetup of Spark Belgum
http://www.meetup.com/Spark-Belgium/events/222632697/

ORDINA CONNECTIVATE

Agenda

Spark Belgium Meetup group introduction

• Andy Petrella (7h 15) Spark Machine Learning using the Spark Notebook

• Toni Verbeiren (8:15 pm) - Connecting Data Scientists to the data (using zeppelin) and

Turning (big) data analytics into a product

- Bridging the gap between the notebook and the interface

• Gerard Maas (9 pm) "Live report" from our reporter Gerard Maas at the Spark Summit in San Francisco bringing you the latest news about Spark.

Want More?

Upcoming **meetup** hosted @ordina on wednesday 24/06 - Neo4j

http://www.meetup.com/graphdb-belgium/events/222504421

ORDINA CONNECTIVATE

• Intro to Graphs: we will present the basics of graph theory and graph databases, and illustrate that some interesting demonstrations.

• Graph case study: we will have a user of Neo4j present their case at the event, their technical and business reasons for adopting Neo4j, and their demonstrated results.

Even More?

Upcoming **workshop** on 2/7 - @ Ordina

Introduction to Hadoop and it's zoo

Questions or Suggestions?

ORDINA

CONNECTIVATE