Big Data
12. Graph Databases
Why graph databases?
The NoSQL paradigms

Key-value stores

Column stores

Triple stores

Document stores
Relational databases...

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Relational databases...
Relational databases... have expensive joins!
Relational databases...

... are not that efficient at relationships!
We already know how to partly solve this though

3NF

0NF
We already know how to partly solve this though

... but it has its limits, too!
Traversals...
Traversals...

... translate into multiple joins!
Reverse traversals...
Reverse traversals...

... need even more indices!
Traversals...

what if links would be more "direct"?

...translate into multiple joins!
Index-free adjacency
Graphs
Graphs: ingredients

Nodes

Edges
Graphs: nodes
Graphs: edges
Graphs: directed graph
Graphs: undirected graph
Graph representation: adjacency list

<table>
<thead>
<tr>
<th>Node</th>
<th>Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>[ ]</td>
</tr>
<tr>
<td>B</td>
<td>[ A, C ]</td>
</tr>
<tr>
<td>C</td>
<td>[ A ]</td>
</tr>
</tbody>
</table>
Graph representation: adjacency matrix
Graph representation: incidence matrix

<table>
<thead>
<tr>
<th>Edges</th>
<th>Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>A</strong></td>
<td>1</td>
</tr>
<tr>
<td><strong>B</strong></td>
<td>-1</td>
</tr>
<tr>
<td><strong>C</strong></td>
<td>0</td>
</tr>
</tbody>
</table>
Labeled property graphs: ingredients
Property graph
Labeled graph
Labels on nodes
Names on relationships
Labeled property graph
Node with properties and label

Name: Einstein
First name: Albert
Profession: Physicist

Person
In Switzerland
Graph database
Graph databases: families

Property Graph

Triple stores (RDF)
Graph databases: native or not

Native Graph Database

<table>
<thead>
<tr>
<th>Source</th>
<th>Target</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Bob</td>
<td>knows</td>
</tr>
<tr>
<td>Eve</td>
<td>Bob</td>
<td>eavesdrop</td>
</tr>
<tr>
<td>Eve</td>
<td>Alice</td>
<td>eavesdrop</td>
</tr>
</tbody>
</table>

Graph stored as RDBMS, document store, ...
RDF
Triple-based graph
RDF: one triple

ETH Zürich Is located in Switzerland

Subject Property Object
IRI

http://www.ethz.ch/#school

http://www.example.com/Switzerland
Literal

Foo  2012-12-16

3.1415926535

includes XML Schema types!
Blank Node

ETH Zürich

Is built on ground

Is subset of

Switzerland
What can appear where?

<table>
<thead>
<tr>
<th></th>
<th>Subject</th>
<th>Property</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>IRI</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Literal</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Blank node</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Generalized Graphs

<table>
<thead>
<tr>
<th></th>
<th>Subject</th>
<th>Property</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>IRI</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Literal</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Blank node</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Syntax
RDF Formats

- RDF/XML
- Turtle
- JSON-LD
- RDFa
- N-Triples
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:geo="http://www.example.com/geography#">
  <rdf:Description rdf:about="http://www.ethz.ch/#self">
    <geo:isLocatedIn
      rdf:resource="http://www.example.com/Switzerland"/>
    <geo:population>8000000</geo:population>
  </rdf:Description>
</rdf:RDF>
RDF/XML: Subject

```xml
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:geo="http://www.example.com/geography#">
  <rdf:Description rdf:about="http://www.ethz.ch/#self">
    <geo:isLocatedIn rdf:resource="http://www.example.com/Switzerland"/>
    <geo:population>8000000</geo:population>
  </rdf:Description>
</rdf:RDF>
```
RDF/XML: Property

```xml
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:geo="http://www.example.com/geography#">
  <rdf:Description rdf:about="http://www.ethz.ch/#self">
    <geo:isLocatedIn
      rdf:resource="http://www.example.com/Switzerland"/>
    <geo:population>8000000</geo:population>
  </rdf:Description>
</rdf:RDF>
```
RDF/XML: Object

<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:geo="http://www.example.com/geography#">
  <rdf:Description rdf:about="http://www.ethz.ch/#self">
    <geo:isLocatedIn rdf:resource="http://www.example.com/Switzerland"/>
    <geo:population>8000000</geo:population>
  </rdf:Description>
</rdf:RDF>
RDF/XML

```xml
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:geo="http://www.example.com/geography#">
  <rdf:Description rdf:about="http://www.ethz.ch/#self">
    <geo:isLocatedIn rdf:resource="http://www.example.com/Switzerland"/>
    <geo:population>8000000</geo:population>
  </rdf:Description>
</rdf:RDF>
```
RDF/XML

```xml
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:geo="http://www.example.com/geography#">
  <rdf:Description rdf:about="http://www.ethz.ch/#self">
    <geo:isLocatedIn
      rdf:resource="http://www.example.com/Switzerland"/>
    <rdf:type
      rdf:resource="http://www.example.com/geography#school"/>
    <geo:population>8000000</geo:population>
  </rdf:Description>
</rdf:RDF>
```
RDF/XML

```xml
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:geo="http://www.example.com/geography#">
  <rdf:Description rdf:about="http://www.ethz.ch/#self">
    <geo:isLocatedIn
       rdf:resource="http://www.example.com/Switzerland"/>
    <rdf:type
       rdf:resource="http://www.example.com/geography#school"/>
    <geo:population>8000000</geo:population>
  </rdf:Description>
</rdf:RDF>
```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:geo="http://www.example.com/geometry#">
    <geo:school rdf:about="http://www.ethz.ch/#self">
        <geo:isLocatedIn rdf:resource="http://www.example.com/Switzerland"/>
        <geo:population>8000000</geo:population>
    </geo:school>
</rdf:RDF>
JSON-LD

{
  "@context": {
    "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
    "geo": "http://www.example.com/geography#"
  },
  "@id" : "http://www.ethz.ch/#self",
  "rdf:type": "geo:school",
  "geo:isLocatedIn": "http://www.example.com/Switzerland",
  "geo:population" : 8000000
}

Turtle

@prefix geo: <http://www.example.com/geography#> .
@prefix countries: <http://www.example.com/> .
@prefix eth: <http://www.ethz.ch/> .

eth:self geo:isLocated countries:Switzerland .
eth:self geo:population 8000000 .
Turtle

@prefix geo: <http://www.example.com/geography#> .
@prefix countries: <http://www.example.com/> .
@prefix eth: <http://www.ethz.ch/> .

eth:self geo:isLocated countries:Switzerland ;
geo:population 8000000 .
Turtle

@prefix geo: <http://www.example.com/geography#> .
@prefix countries: <http://www.example.com/> .
@prefix eth: <http://www.ethz.ch/> .

eth:self geo:isLocated countries:Switzerland, countries:Europe ;
geo:population 8000000 .
Querying
Querying paradigms

Classical declarative querying

Query by example
Two languages

Cypher

SPARQL
Two languages

Cypher

SPARQL
Querying labeled property graphs by example
Querying labeled property graphs by example
Querying labeled property graphs by example
Querying labeled property graphs by example
Querying labeled property graphs by example
Querying labeled property graphs by example
Cypher pattern

(alpha)-[:A]->(beta)-[:B]->(gamma)
Cypher pattern: anchoring a label

(alpha)  
-[:A]-(beta:yellow)  
-[:B]-(gamma)
Cypher pattern: filtering a property

(alpha {name: 'Einstein'})
-[:A]->(beta)
-[:B]->(gamma)
Cypher pattern: anchoring and filtering

(name: ETH)

(alpha)
-[:A]->(beta)
-[:B]->(gamma: blue {name: 'ETH'})
Cypher pattern: right to left

(alpha)
-[:A]->(beta)
-[:B]->(gamma)
<-[[:B]]-(delta)
Cypher pattern: variable repetition

(alpha)
-[:A]->(beta)
-[:B]->(gamma)
<-[:B]-(delta)
-[:B]->(alpha)
Cypher pattern: variable length path

(\text{alpha}) -[*1..4]\rightarrow (\text{beta})
Cypher pattern: MATCH clause

(alpha {name: 'Einstein' })-[:A]->(beta)-[:B]->(gamma)
Cypher pattern: MATCH clause

```cypher
MATCH (alpha {name: 'Einstein' })-[A]->(beta)-[B]->(gamma)
RETURN gamma
```
Cypher pattern: WHERE clause

MATCH (alpha {name: 'Einstein' })-[[:A]]->(beta)-[:B]->(gamma)
RETURN gamma

MATCH (alpha)-[:A]->(beta)-[:B]->(gamma)
WHERE alpha.name = 'Einstein'
RETURN gamma
Cypher pattern: CREATE clause

```
CREATE (einstein:Scientist {name: 'Einstein', first: 'Albert' }),
(eth:University {name: 'ETH Zurich' }),
(einstein)-[:VISITED]->(eth)
```
Other clauses

WITH
DELETE
SET
UNION
START
MERGE
FOREACH
Two languages

Cypher

SPARQL
Querying RDF: SPARQL

PREFIX geo: <http://www.example.com/geography#>
PREFIX countries: <http://www.example.com/>

SELECT ?s
WHERE { ?s geo:isLocatedIn countries:Switzerland }
PREFIX geo:  <http://www.example.com/geography#>
PREFIX countries:  <http://www.example.com/>

SELECT  ?s
WHERE  {  ?s geo:isLocatedIn countries:Switzerland  }
PREFIX geo: <http://www.example.com/geography#>
PREFIX countries: <http://www.example.com/>

SELECT ?s
WHERE { ?s geo:isLocatedIn countries:Switzerland }
PREFIX geo: <http://www.example.com/geography#>
PREFIX countries: <http://www.example.com/>

SELECT ?s
WHERE {
  ?s geo:isLocatedIn countries:Switzerland .
  ?s :deliversDiplom :bachelor .
}
PREFIX geo: <http://www.example.com/geography#>
PREFIX countries: <http://www.example.com/>

SELECT ?s
WHERE {
  ?s geo:isLocatedIn ?c .
  ?c geo:isInContinent geo:America .
}
PREFIX geo: <http://www.example.com/geography#>
PREFIX countries: <http://www.example.com/>

SELECT ?s
WHERE {
  ?s geo:isLocatedIn countries:Switzerland.
  ?s :deliversDiplom :bachelor
}
LIMIT 10
PREFIX geo: <http://www.example.com/geography#>
PREFIX countries: <http://www.example.com/>

SELECT ?s ?name
WHERE {
  ?s geo:isLocatedIn countries:Switzerland .
  ?s :deliversDiplom :bachelor .
  ?s :hasName ?name .
}
ORDER BY ?name
LIMIT 10
Architecture (Neo4j)
No sharding
Document stores don't like joins.

Graph databases don't like shards.
Why? Fast traversal
Master-slave architecture
Data replication

- Master
- Slave

Diagram shows a master node connected to three slave nodes.
Data replication

Master

Slave

Slave

Slave

Synchronization
Data replication (full)
Read scale-up
Writes

Write to the master

or

Write to a slave
Caching and pages

Index-free adjacency

Fixed-size records
Label storage

Person
Jedi
Geek

Person → Jedi → Geek
Properties storage

name: Einstein
first-name: Albert
Relationship storage
Relationship storage
Relationship storage
Relationship storage
Relationship storage
Relationship storage
Relationship storage
Relationship storage
Relationship storage
Relationship storage
Relationship storage
Relationship storage
Relationship storage
Relationship storage
Relationship storage
Relationship storage

A

B

s-previous
s-next
t-previous
t-next

Source
Target
Typical sizes

Node: 9 bytes
Relationship: 33 bytes
Relationship name: 5 bytes
Property: 33 bytes
RDF has no semantics
RDF Schema

Class

Property
Classes

rdfs:Resource
rdfs:Class
rdf:Property

rdfs:Literal
rdfs:DataType
rdf:HTML
rdf:XMLLiteral
Properties

On any resources
- rdf:type
- rdfs:label
- rdfs:comment

On properties
- rdfs:range
- rdfs:domain
- rdfs:subPropertyOf

On classes
- rdfs:subClassOf
Self-awareness

rdfs:Resource \(\xrightarrow{\text{rdf:type}}\) rdfs:Resource
Self-awareness
Self-awareness

```
self-awareness rdf:type rdfs:Class
```

```
self-awareness rdf:range rdfs:Class
```
Self-awareness

```
rdf:subClassOf
```

```
rdf:type
```

```
rdfs:Property
```
Simple Entailment (RDF semantics)

$I(E) = \text{true}$

$I$
OWL

**Definition of ONTOLOGY**

1. a branch of metaphysics concerned with the nature and relations of being
2. a particular theory about the nature of being or the kinds of things that have existence
OWL

(In principle) **standalone**

(Much) More **powerful** than RDF(S)
OWL
OWL and description logic / AI
Entailment (and Syllogisms)

Major  All men are mortal.
Minor  Socrates is a man.

Conclusion  Therefore, Socrates is mortal.
Trees...
... and Graphs