

graphdb++

11.06.2014

Valerij Frühauf

Sascha Klein

Marc Frerichs

Agenda

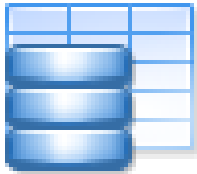
- Was ist graphdb++?
- Funktionalität
- Beispiele
- Aussicht

Was ist graphdb++?

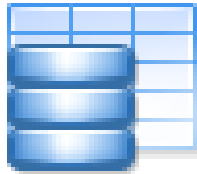
- Mapper: ein Stück Software zum Datenaustausch zwischen zwei Komponenten
- RDBMS \Leftrightarrow Graphrepräsentation
- Funktionen zur Persistierung von Graphen (CRUD: **c**reate, **r**ead, **u**pdate, **d**eleate)
- Graphtraversierung

Was ist graphdb++?

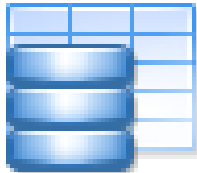
NODE



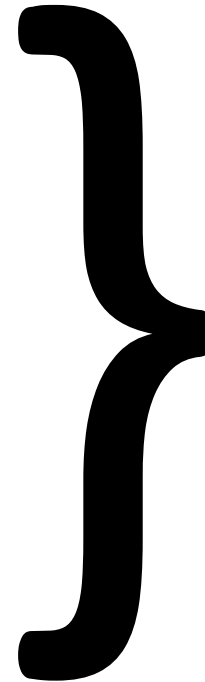
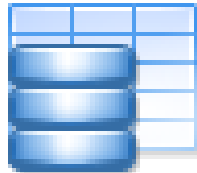
EDGE



NODE_PROPERTIES



EDGE_PROPERTIES



Funktionalität

Klassen

- GraphDB
- GraphElement
 - Node
 - Edge
- Traverser
- Result
- Track

Sonst

- DBUtil.h
- Types.h
- Visualization.h

Funktionalität

- GraphDB
 - erzeugt neue bzw. nutzt vorhandene GraphDB
 - getAllNodes / getAllEdges
 - getNodeById / getEdgeById
 - deleteDB / clearDB
 - getTypedNodes / getTypedEdges
 - getPropertyNodes / getPropertyEdges

Funktionalität

- GraphElement
 - abstrakte Klasse
 - implementiert gemeinsame Funktionen von Node und Edge
 - getId
 - getType
 - deklariert virtuelle Funktionen
 - getProperty / setProperty / deleteProperty
 - getProperties / clearProperties
 - remove

Funktionalität

- Node
 - getEdges (OUTGOING, INCOMING, ALL)
- Edge
 - getSource / getTarget

Node und Edge implementieren weiterhin die genannten Funktionen von GraphElement.

Funktionalität

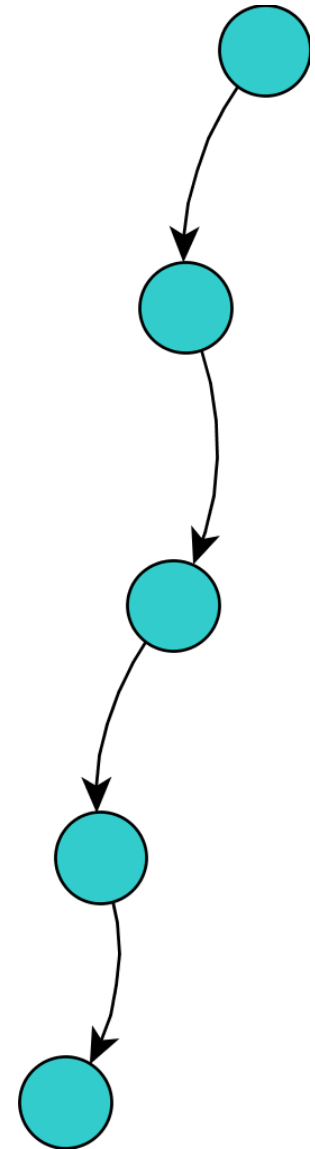
- Traverser
 - „Durchlaufen“ des Graphen; ausgehend von einem bestimmten Knoten
 - setDirection (OUTGOING, INCOMING, ALL)
 - setKeepGoing (true/false)
 - addEdgeType / addNodeType
 - addEdgeProperty / addNodeProperty
 - depthFirst (liefert Result)
 - paths (liefert vector<Track>)

Funktionalität

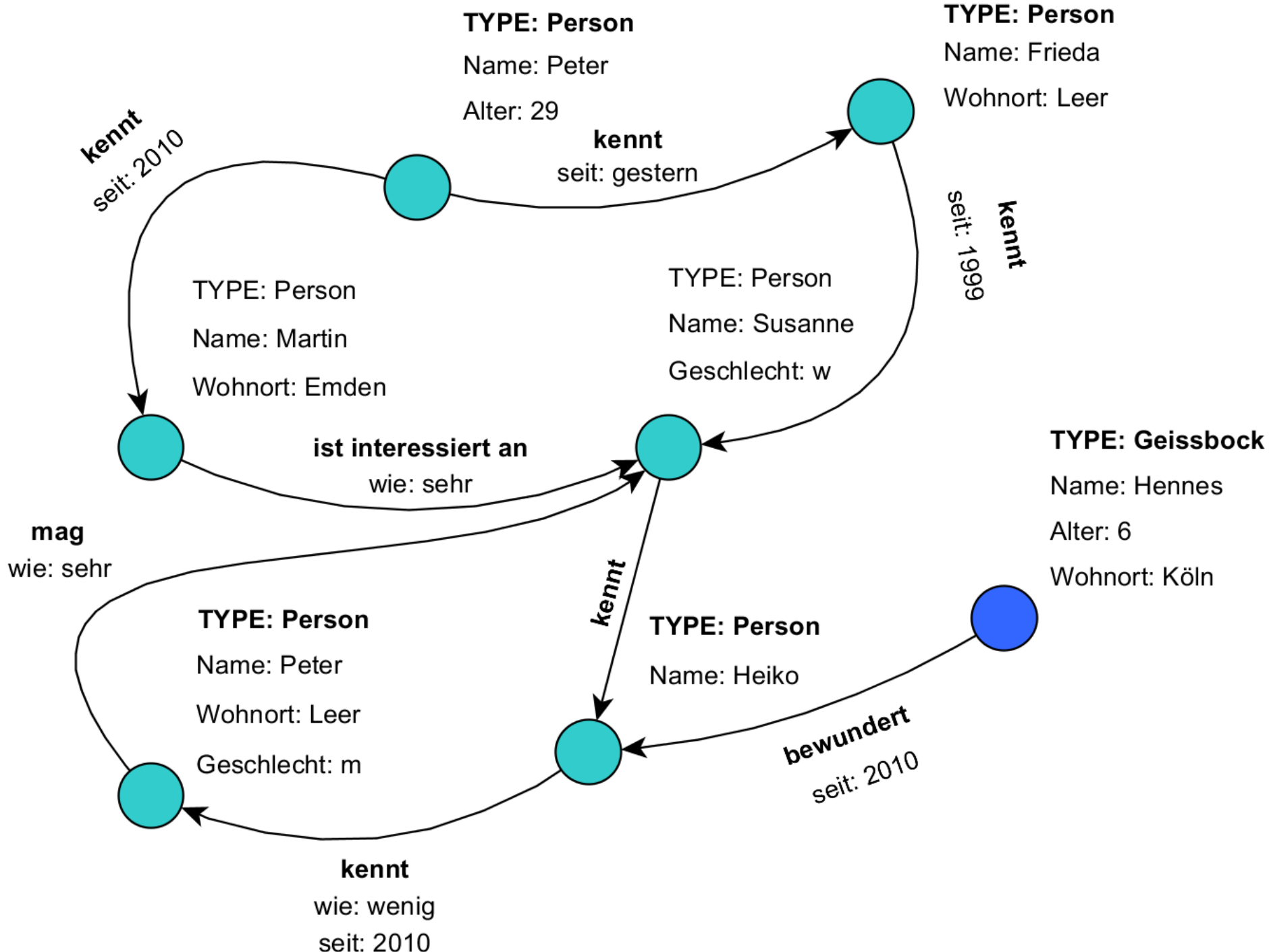
- Result
 - Ergebnismenge der Graphtraversierung
 - Knoten
 - Kanten
 - `getNodes` / `getEdges` / `getUniqueEdges`
 - `print` (benötigt `GraphViz`)

Funktionalität

- Track
 - Pfad
 - abwechselnde Folge von Knoten und Kanten
 - getPath
(liefert `vector<GraphElement*>`)



Beispiele



Beispiele

```
int main() {  
    GraphDB gdb("soziales_netzwerk");           // GraphDB erzeugen  
    Node peter(gdb, "Person");                  // Knoten erzeugen  
    peter.setProperty("Name", "Peter");         // Property setzen  
    ...  
    Node hennes(gdb, "Geissbock");              // anderer Knotentyp  
    ...  
    // Kante erzeugen:  
    Edge peter_kennt_frieda(gdb, peter, frieda, "kennt");  
    peter_kennt_frieda.setProperty("seit", "gestern");  
    ...  
}
```

Beispiele

```
int main() {
```

```
...
```

```
Traverser t(peter);
```

```
t.addEdgeType("kennt");
```

```
t.addNodeType("Person");
```

```
t.setKeepGoing(false);
```

```
Result r = t.depthFirst();
```

```
// Traverser erzeugen
```

```
// Kanten einschränken
```

```
// Knoten einschränken
```

```
// nicht „weiterlaufen“
```

```
// Traversierung starten
```

```
vector<Node> vn = r.getNodes();
```

```
vector<Edge> ve = r.getEdges();
```

```
...
```

Beispiele

```
int main() {  
    ...  
    vector<Track> vt = t.paths(frieda, heiko);           // Weg(e) von Frieda zu Heiko  
  
    for(vector<Track>::iterator it = vt.begin(); it != vt.end(); it++) {  
        vector<GraphElement*> vge = it->getPath();  
        for(vector<GraphElement*>::iterator gelt = vge.begin(); gelt != vge.end(); gelt++) {  
            ...  
        }  
        ...  
    }  
    ...  
}
```


Aussicht

- Grundfunktionalität gegeben
- kann/sollte noch erweitert werden

Bei Interesse kann zum SVN-Repository eingeladen werden.

