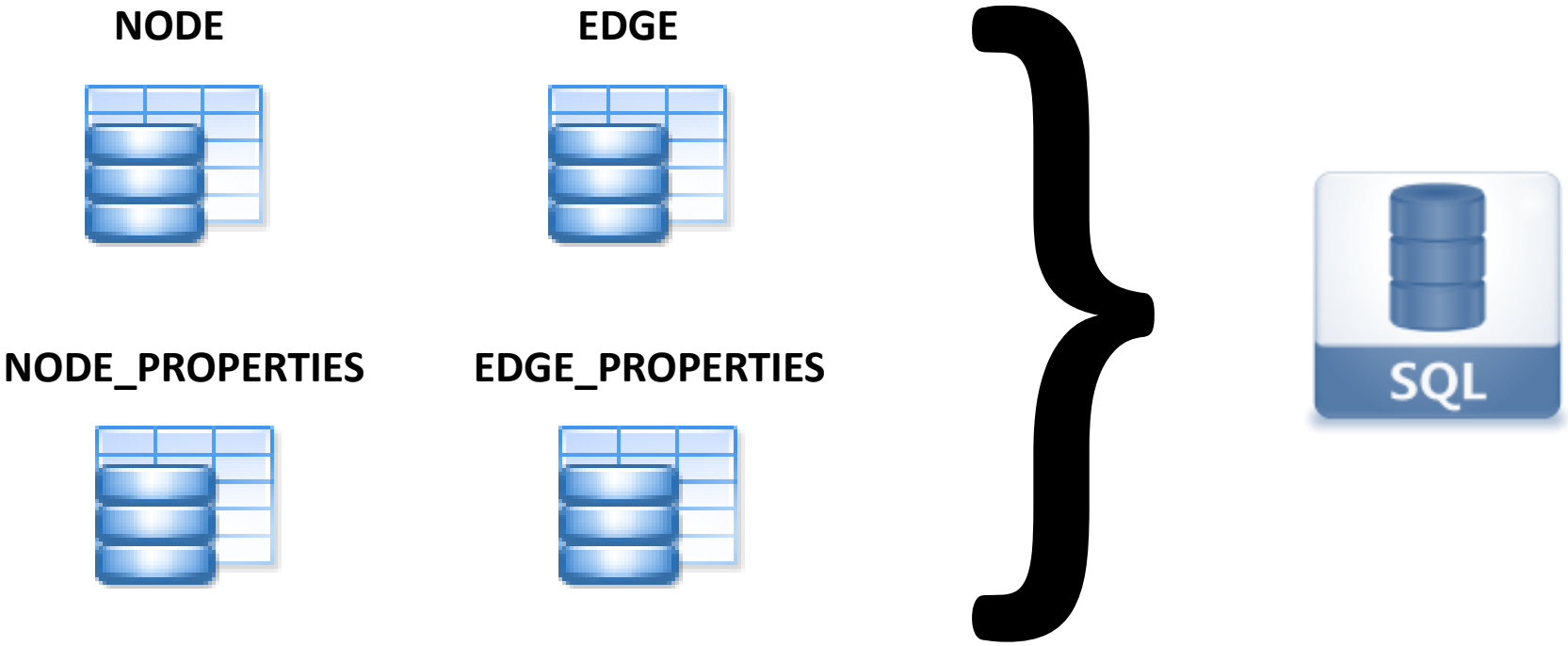
Was ist graphdb++?

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

Was ist graphdb++?



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>)

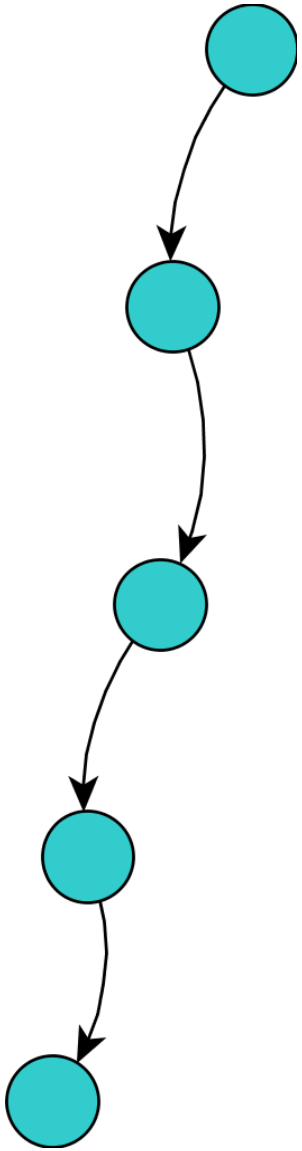
Funktionalität

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

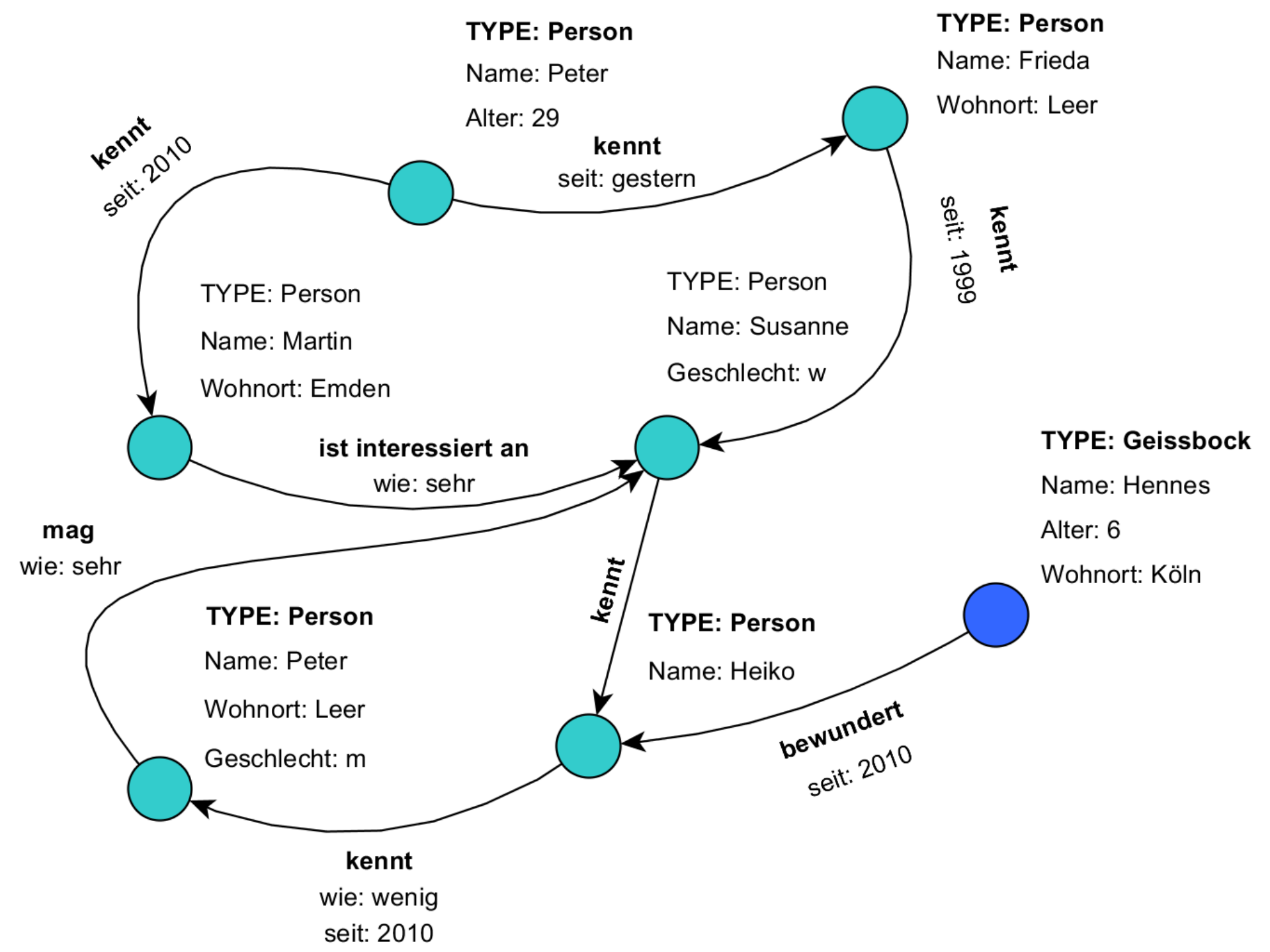
Funktionalität

- Track
  - Pfad
  - abwechselnde Folge von Knoten und Kanten
  - getPath

    (liefert vector<GraphElement*>)



Beispiele


TYPE: Person
Name: Peter
Alter: 29
TYPE: Person
Name: Frieda
Wohnort: Leer
kennt
seit: 2010
kennt
seit: gestern
kennt
seit: 1999
TYPE: Person
Name: Martin
Wohnort: Emden
TYPE: Person
Name: Susanne
Geschlecht: w
ist interessiert an
wie: sehr
TYPE: Geissbock
Name: Hennes
Alter: 6
Wohnort: Köln
mag
wie: sehr
kennt
TYPE: Person
Name: Peter
Wohnort: Leer
Geschlecht: m
TYPE: Person
Name: Heiko
bewundert
seit: 2010
kennt
wie: wenig
seit: 2010

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);                    // Traverser erzeugen
t.addEdgeType("kennt");                // Kanten einschränken
t.addNodeType("Person");               // Knoten einschränken
t.setKeepGoing(false);                 // nicht „weiterlaufen“
Result r = t.depthFirst();             // 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 geIt = vge.begin(); geIt !=  vge.end(); geIt++) {
      …
   }
   …
}
…
```