Dresden Database Systems Group

Großer Beleg

Entwicklung eines Frontends zur Durchführung von Nutzerstudien im Umfeld von „Why“-Graphanfragen

Christoph Scheidig

Gliederung

- Einleitung
- Lösungsansätze
- Meine Implementierung

Einführung

- Vorteile von Graphdatenbanken:

  - Flexibilität, Ausdrucksstärke von Anfragen

  - stark vernetzte Informationen, strukturierte Zusammenhänge

- Nachteil: Über-/ Unterspezifikation von Anfragen

  → zu wenige, zu viele, fehlende Ergebnisse

  → systematische Umschreibung

- Ziel: einheitliche Benutzeroberfläche für Nutzerstudien

Grundlagen

Dresden Database Systems Group

- Attributgraph (V, E, u, f, g, A_V, A_E)
- „Why so few?“
- „Why so many?“
- „Why empty?“
- („Why not?“)
- Vor allem „why empty?“ und „why so many?“

Lösungen

Dresden Database Systems Group

- Teilgraphbasiert vs. Umschreibung der Anfrage
- Einfache Operationen (entgegengesetzt):

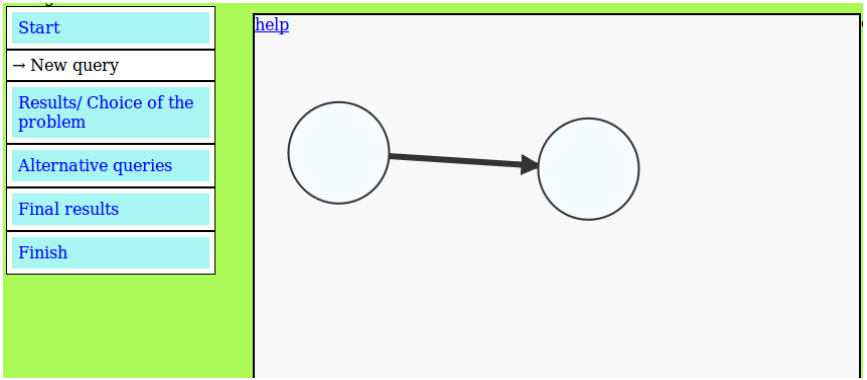
  - topologisch: Knoten, Kanten, Richtungen, Quellen, Ziele löschen/ einfügen

  - semantisch: Operatoren, Prädikate, Konstanten, Typen
- Komplexe Operationen

Die Implementierung

Dresden Database Systems Group

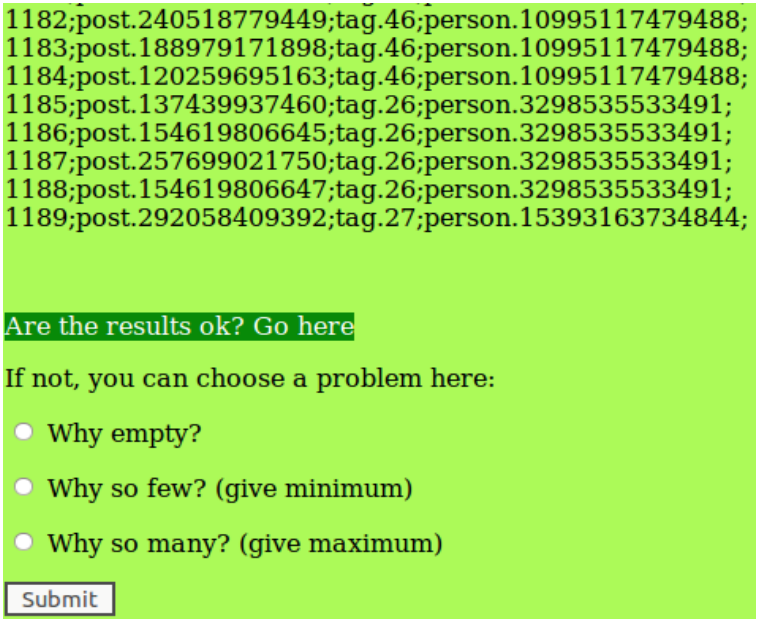
- Manuelles Erstellen von Anfragen
  - graph. Editor
  - Auswahl aus einigen Vorlagen
- GUI mit Klick für Knoten und Kanten
- Auswahlliste für Attribute
- Zeit läuft seitenübergreifend mit



Die Implementierung

Dresden Database Systems Group

- Abschicken der Anfrage
- Abwarten der Ergebnisse, textuelle Anzeige
- Wahl des Problems, Radio-Button, ggf. Eingabe der Kardinalität
- Warten, Anzeige alternativer Anfragen mit Radio-Button
- Bei Auswahl deren Ausführung



Die Implementierung

Dresden Database Systems Group

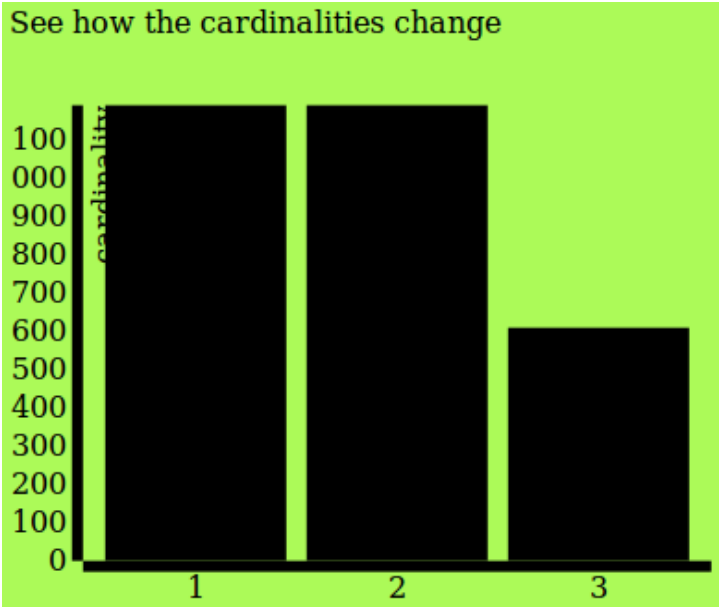
- Zurückspringen

  - Ändern der Anfrage

  - Rückkehr zu Ergebnissen oder Alternativen

- Speichern der Knoten/ Kanten als sessionStorage
  - Konvertierung nötig

- Säulendiagramm: Abfolge der Kardinalitäten



Node.js

Dresden Database Systems Group

- Plattform zum Betrieb von Netzwerkanwendungen und insbes. lokale Webserver
- JavaScript definiert ereignisgesteuerte Architektur
- Ausführungsumgebung kompiliert JS-Code in Maschinencode
- Aufruf über **http://localhost:Port**
- Erweiterung durch express – Verzeichnisstruktur

Node.js

Dresden Database Systems Group

```
var express = require('express'); var app = express();
var server = require('http').createServer(app);
server.listen(8080);
app.get('/', function(req, res{
  fs.writeFileSync('/home/osboxes/Documents/Beleg/data.tsv',
"number\tamount");
  res.sendFile('/home/osboxes/Documents/Beleg/index.html');
});
```

Socket.io

- Zusätzl. Paket im Server, JS-Bibliothek im Client
- Asynchrone Kommunikation zwischen Client und Server
- Übergabe von Anfrage etc. an Server
- Anfrage nach den Ergebnissen und deren Übergabe

Verbindung zum Server

Dresden Database Systems Group

- Nach Laden zählt Client Sekunden, Anfrage *socket.emit('requestdata',{})*
- Einlesen der Dateien mit Alternativen/ Ergebnissen: *fs.readFileSync*
- *givedata*-Nachricht an Client, Daten als Variable
- Dort: *socket.on('givedata',...){Verarbeitung und Anzeige: data.variable}*
- Datenbankserver läuft als Kindsprozess, Schreiben in stdin-Port

Das Diagramm

Dresden Database Systems Group

- Ablage der Kardinalitäten durchgeführter Anfragen in data.tsv

- Darstellung durch **d3.js**: JS-Bibliothek zur DOM-basierten, dynamischen, interaktiven Visualisierung von Daten durch Diagramme, Graphen,...

  - SVG-Datei, Manipulation über DOM-Knoten

Quellen

E. Vasilyeva, M. Thiele, C. Bornhövd, W. Lehner, Answering “Why Empty?” and “Why So Many?” queries in graph databases, Journal of Computer and System Sciences 82 (2016), Elsevier, 2015

http://bl.ocks.org/mbostock/3885304

http://bl.ocks.org/cjrd/6863459