Gráfadatbázisok

Tanulmány az Adatbázisok haladóknak c. tárgyhoz

Szárnyas Gábor

5. évf. mérnök informatikus szak

2012/2013. tanév I. félév

Tartalomjegyzék

Bevezetés

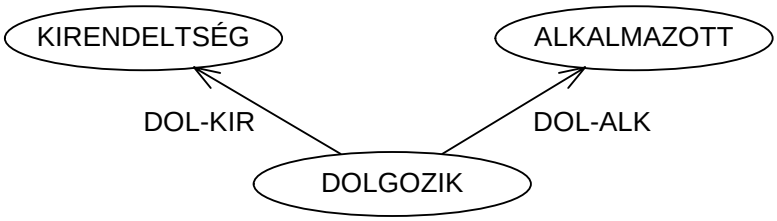
Komplex, sok összefüggést tartalmazó adathalmazokat gyakran célszerű gráffal reprezentálni. A manapság legelterjedtebb relációs adatbázisok azonban csak korlátozottan alkalmasak gráfok tárolására, ugyanis a gráfokon végzett műveletek – bejárás, tranzitív lezárt számítása – általában költséges illesztés műveletekkel jár.
A *gráfadatbázisok* (graph database) gráfok hatékony tárolását és ezáltal gráfműveletek gyors végrehajtását teszik lehetővé.

Előzmények

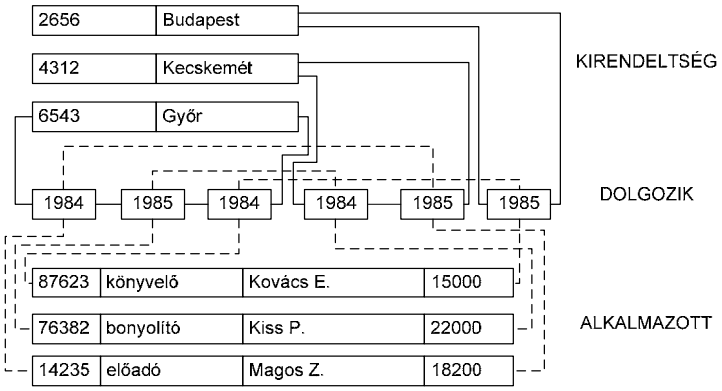
Bár nagyon sok valós szerkezet modellezhető természetes módon gráfokkal, pár évvel ezelőttig csak akadémiai és tudományos körökben alkalmaztak gráfadatbázisokat. A NoSQL adatbázis-kezelők között azonban reneszánszukat élik a gráf adatmodellt használó rendszerek.
A gráfadatbázisok elődeinek tekinthetők a hálós adatbázisok és a triplestore-ok.

Hálós adatbázisok

A *hálós adatbázisok* (network database) esetén – a gráfadatbázisokkal ellentétben – az adatbázis sémája előre definiált [1]. A *sémát* egy gráffal ábrázoljuk, melyben a csomópontok (set típusok) az entitásokat, az élek (rekord típusok) az entitások kapcsolatait ábrázolják. Míg a hálós adatbázisok strukturáltan tárolják az adatokat (erre szolgálnak a rekord típusok és a set típusok), a gráfadatbázisokban nincs előre definiált séma.



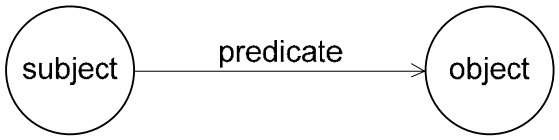
1. ábra: Hálós adatbázis sémája. Az csomópontok a rekord típusok, az élek a set típusok.



2. ábra: Az 1. ábra sémájára illeszkedő hálós adatbázis

Triplestore-ok

A triplestore-ok olyan specializált adatbázis-kezelők, amelyek a *Resource Description Frameworkben* (RDF) meghatározott *hármasokat* (triple) tárolnak. A triple-ök állításokat fogalmaznak meg *tárgy–predikátum–objektum* (subject-predicate-object) formában. Az RDF a W3C 1999-es ajánlása.



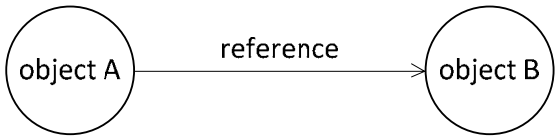
3. ábra: Triple

4. ábra: RDF adatmodell

A triplestore-ok adatmodelljében az éleknek nincsenek tulajdonságai (pl. súly). A triplestore-ok és a gráfadatbázisok között fontos különbség, hogy a triplestore-ok fő célja ún. következtetési algoritmusok (*reasoning*) futtatása, míg a gráfadatbázisok esetén tipikusan (részgráf) mintaillesztéseket (*graph pattern matching*) és útbejárásokat (*graph traversal*) végzünk.

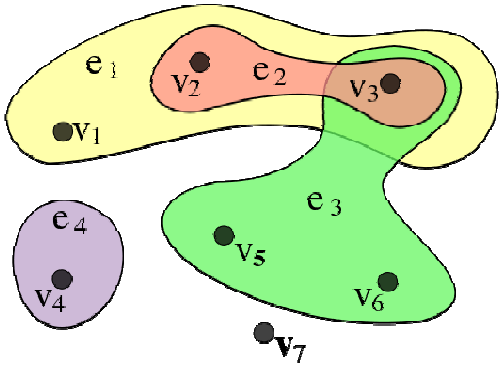
Objektum-orientált adatbázis-kezelők

Bár a gráfadatbázisokhoz csak lazán kapcsolódnak, megemlítjük a témához lazán kapcsolódó objektum-orientált adatbázis-kezelőket is. Az objektum-orientált adatmodellben az adatmodell (vizuálisan *osztálydiagram*) és a futás során példányosított objektumok (vizuálisan *objektumdiagram*) is reprezentálhatók gráfként.



5. ábra: Objektum-orientált adatbázis

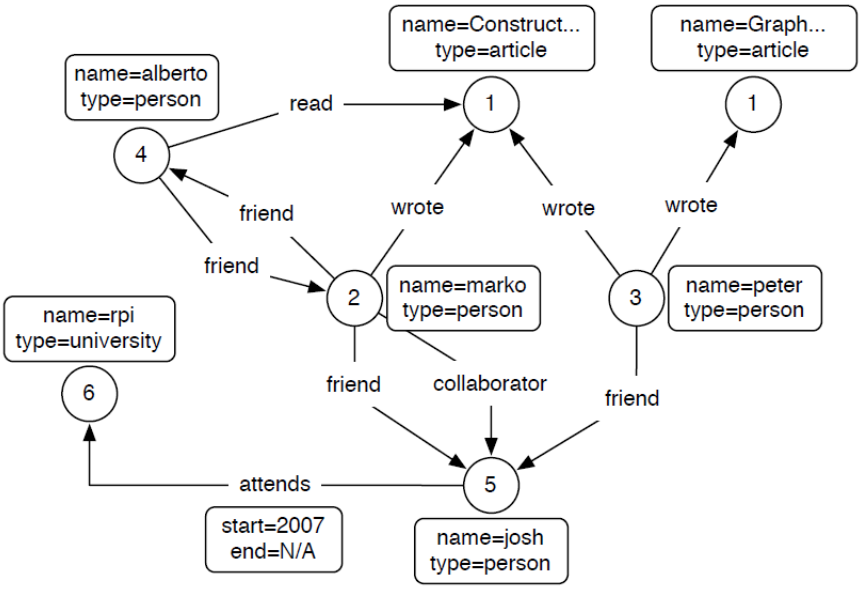
Adatmodell



6. ábra: Irányítatlan hipergráf

A gráf adatmodellnek sokféle változata létezik, ezekből néhány:

- *egyszerű gráf* (simple graph): csúcsok halmaza (V) és élek halmaza (E), $E \subseteq \binom{V}{2}$.[1]
- *multigráf* (multi-graph): két csomópont között több él is futhat.
- *irányított gráf* (directed graph): az élek irányítottak ($E \subseteq V \times V$).
- *súlyozott gráf* (weighted graph): az élekhez súlyt rendelünk.
- *hipergráf* (hypergraph): egy él több csomópontot is érinthet (6. ábra).[2]
- *szemantikus gráf* (semantic graph): subject–predicate–object hármasokat tartalmaznak.
- *tulajdonsággráf* (property graph): olyan irányított multigráf, amelyben az éleknek típusa van és az egyes csomópontokhoz és élekhez kulcs-érték párokat rendelhetünk (7. ábra).



7. ábra: Tulajdonsággráf [6]

[1] Gyakran az *egyszerű gráf* definíciója kiköti azt is, hogy a gráf nem tartalmazhat hurokéleket. Itt ettől a formális részlettől eltekintünk. A $G(V, E)$ struktúra az egyetemi matematikai tárgyakban gyakori, ezért gyakran „textbook-style” gráfként hivatkoznak rá.

[2] Érdekesség, hogy a relációs adatbázisok elméletében fontos szerepet játszó funkcionális függőségek egy halmaza irányított hipergráffal ábrázolható, ld. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.33.787&rep=rep1&type=pdf

A tulajdonsággráfot általában a $G(V, E, \lambda)$ hármassal jelölik, ahol λ a csúcsokhoz és élekhez tulajdonságokat (kulcs–érték párokat) rendelő függvény.
A gráfadatbázisok jellemzően *tulajdonsággráfokat* (property graph) tárolnak. A TinkerPop keretrendszer (8. ábra) tartalmaz egy Blueprints (9. ábra) nevű tulajdonsággráf interfészt. A Blueprintsben definiált tulajdonsággráf az alábbi módon épül fel:

- csúcsok
  - egyedi azonosító
  - kimenő élek
  - bemenő élek
  - tulajdonságok: kulcs–érték párok
- élek
  - egyedi azonosító
  - forrás csomópont
  - cél csomópont
  - típuscímke
  - tulajdonságok: kulcs–érték párok

8. ábra: A TinkerPop keretrendszer logója

9. ábra: A Blueprints logója

Gráffeldolgozó eszközök

A gráf definíciója – egy gráf csúcsok és élek halmaza – egy nagyon általános, de egyszerű, matematikailag jól elemezhető adatstruktúrát ad. A gráfelmélet az elmúlt évtizedekben intenzív kutatások tárgya volt, melynek során nem csak a matematika többi területével kötötték össze, hanem nagyon sok gyakorlati esetre alkalmazták sikerrel. Ennek megfelelően informatikai rendszerekben is gyakran alkalmaznak gráfokat adatok reprezentációjára és feldolgozására.
Az alábbiakban felsoroljuk a gráffeldolgozó keretrendszerek fő kategóriáit [18].

Gráf keretrendszerek

A gráfokkal dolgozó alkalmazások között az egyik leggyakoribb megközelítés, hogy a fejlesztők az adatstruktúrákat és az algoritmusokat nem implementálják, hanem valamilyen szoftver *könyvtárat* (library) használnak.

Ezek a könyvtárat általában a teljes gráfot a számítógép memóriájában tárolják. A gráf keretrendszer a lefordított alkalmazás része lesz, ezért ezek a rendszerek több felhasználó kezelését és a többi megszokott adatbázis szolgáltatást nem nyújtják.
A gráf keretrendszer tipikusan egyszerű, irányított gráfokon dolgoznak.
Példák:

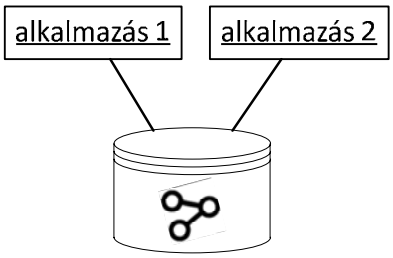
- JUNG (Java Universal Network/Graph) – Java,
- igraph – C nyelv, magyar fejlesztés [17],
- NetworkX – Python.

Gráf keretrendszer és az alkalmazás viszonya

Gráfadatbázisok

A gráfadatbázisok a NoSQL rendszerek egyik fő kategóriáját alkotjuk. Az adatokat általában diszken tárolják és általában csak replikációt támogatnak. Ezek a rendszerek már képesek több felhasználó egyidejű kezelésére, tranzakciók futtatására és konzisztencia biztosítására.
A gráfadatbázisok tipikus adatmodellje a tulajdonsággráf.
Példák:

- Neo4j – Java,
- InfiniteGraph – Java,
- dex – C++.

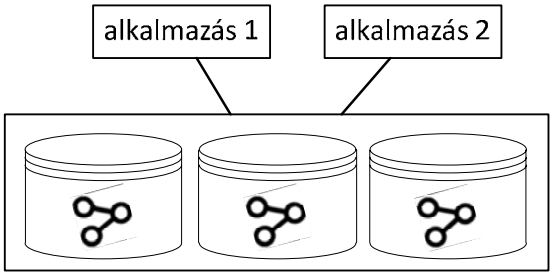


Gráfadatbázis és az alkalmazások viszonya

Gráf fürtök

A Google 2010-ben publikálta a Pregel keretrendszerének alapötletét [16]. A MapReduce-hoz hasonlóan a cikk csak az elméleti ötletet („Bulk Synchronous Parallel Processing") tartalmazza, a Google saját C++ nyelvű implementációja nem nyilvános. A gráf fürtök nagyfokú elosztottságot támogatnak. A gráf mérete roppant nagy lehet, ezt csak a fürt számítógépeinek összmemóriája korlátozza.
A gráf fürtök tipikus adatmodellje az egyszerű gráf.
Példák:

- Hama – Java,
- Giraph – Java,
- GoldenOrb – Java.



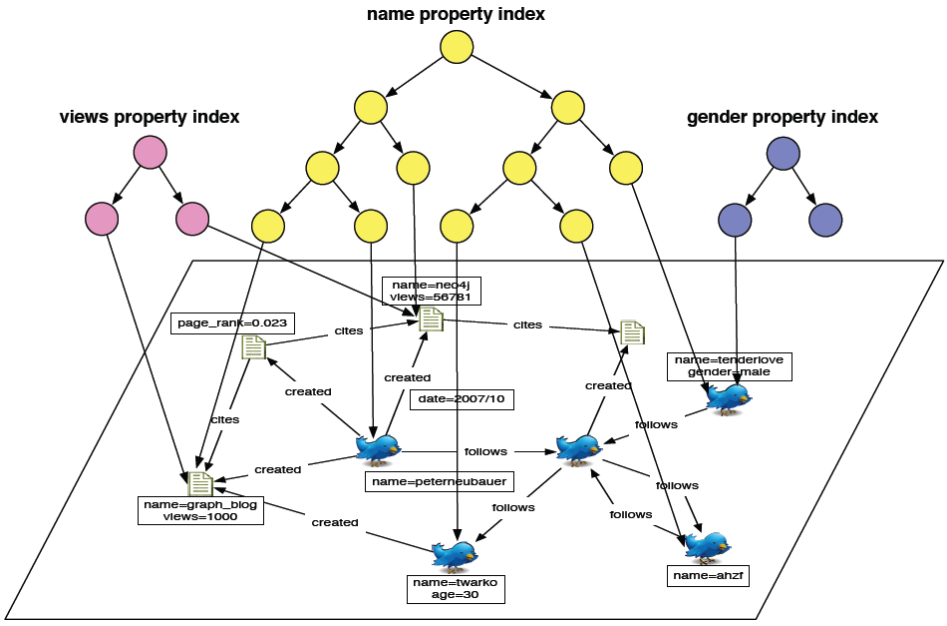
Gráf fürtök és az alkalmazások viszonya

Gráfadatbázisok

A többi NoSQL adatbázis-kezelőhöz hasonlóan a gráfadatbázis-kezelőknek sincs precíz definíciója. Az alábbiakban körvonalazzuk a gráfadatbázisok jellemzőt.
Szinte minden adatbázis-kezelőben tárolhatunk gráfot, pl. a gráf éllistájának tárolásával. A gráfadatbázisokban azonban a gráfot *explicit módon* tároljuk és *minden csomópontnak közvetlen mutatója* van a szomszédos csomópontokra. Így a gráf bejárásához nincs szükség az adatbázisból olvasott adatok gráffá alakítására, táblák illesztésére, indexek használatára stb.
A gráfadatbázisok komoly hátránya, hogy jelenleg kevés kiforrott eszközzel rendelkeznek. Az egységes adatmodell hiánya miatt fennáll a *gyártói függőség* (vendor lock-in) veszélye.

Előnyök

A gráfadatbázisok erőssége akkor mutatkozik meg, ha a mögöttes adatmodell egy gráf jellegű struktúra. Amennyiben a gráfon sok útkeresés jellegű műveletet kell futtatni, a gráfadatbázis több nagyságrenddel gyorsabb lehet a relációs rendszereknél.
A gráfadatbázisok kiválóan alkalmazhatók ajánlórendszerek fejlesztésére, ahol pl. egyes felhasználók vásárlásai alapján kell további termékeket ajánlanunk nekik.
A hagyományos adatbázisok indexelés „natív módon”, a gráf adatszerkezetébe ágyazva elvégezhető (10. ábra). A relációs adatbázis-kezelőkkel szemben itt nem csupán néhány indexstruktúra közül válogathatunk, hanem implementálhatjuk a saját indexünket is, pl. térinformatikai rendszerekhez készíthetünk saját R-fát (R tree, rectangle tree).

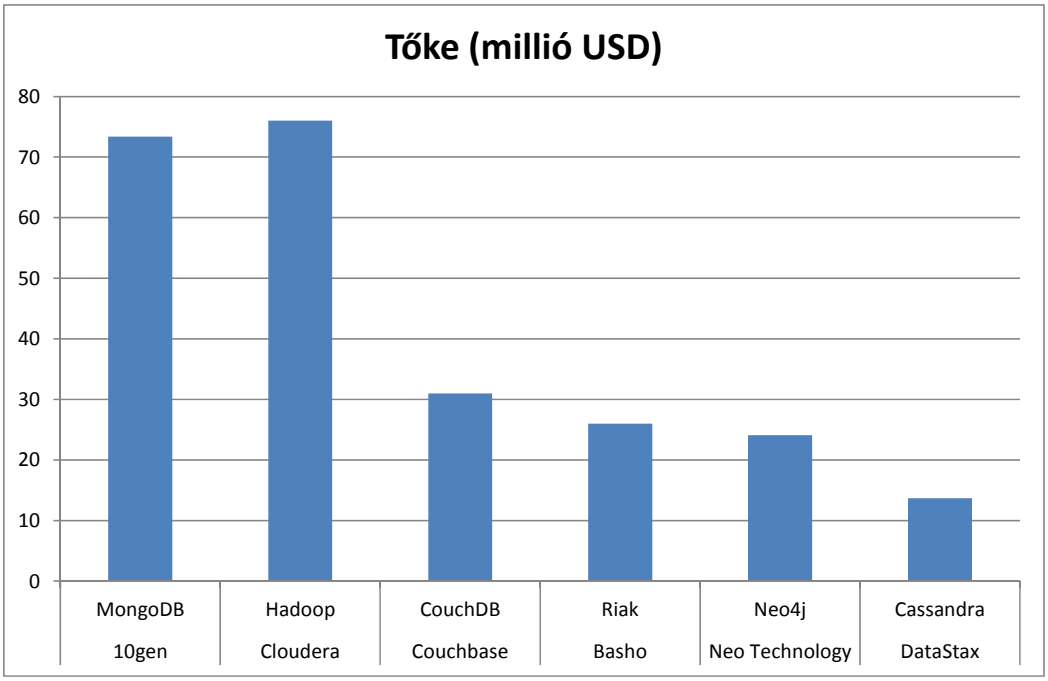


10. ábra: Index struktúra

A tulajdonsággráf adatmodellben a csomópontok kapcsolatainak számára és típusára nincs korlát, ezért *egyetlen gráfban* tárolható az összes adatunk, nincs táblához hasonló struktúra.

A Neo4j rendszer

A svéd Neo Technology cég fejleszti 2002 óta. Az első nyilvános verzió 2007-ben jelent meg. Ez a legnagyobb tőkével rendelkező gráfadatbázis.



11. ábra: NoSQL adatbázisok alaptőkéje

Skálázhatóság

A NoSQL rendszerek között a gráfadatbázisok támogatják legkevésbé a *shardingot*, azaz a gráf csomópontjainak szétosztását az egyes szerverek között. Ennek fő oka, hogy a kulcs-érték tárolókkal és a dokumentumtárolókkal szemen itt explicit kapcsolat van az egyes csomópontok között. A gráf adatmodellel tipikusan sok kapcsolattal rendelkező adathalmazokat reprezentálunk. Míg a kulcs-érték adatbázisoknál általában jó megoldás a kulcs alapján „szétszórni" a kulcs-érték párokat az egyes szerverek között, gráfok esetén már a szétosztás szempontja sem egyértelmű. Gráfbejárási műveletek esetén, ha az egyes csomópontok különböző szerverekhez tartoznak, hálózati kommunikációra van szükség, ami költséges.

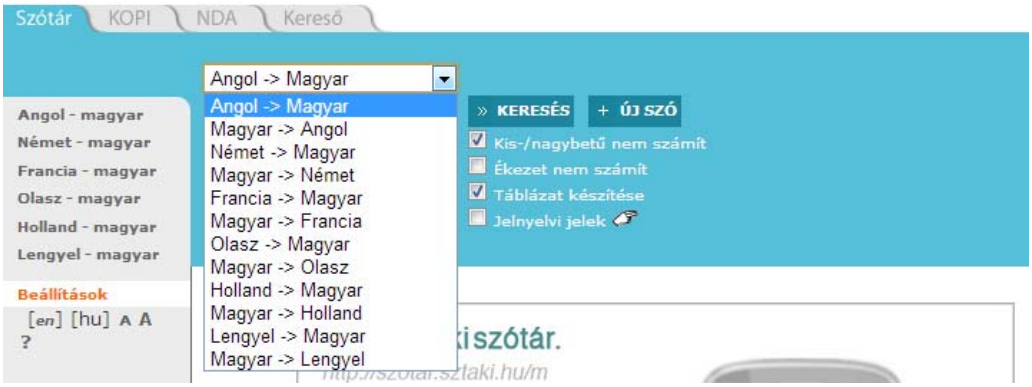
Sharding

A sharding hiánya természetesen komolyan korlátozza a skálázhatóságot, hiszen ennek hiányában csak replikáció valósítható meg, ahol minden számítógépen el kell tárolnunk a teljes gráfot.
Skálázási ötletek [14]:

- Vertikális skálázás
  - Működés: sok memóriát teszünk a szerverbe. A Neo4j gyorsítótárként használja a memóriát, ezért ha az egész adathalmazunk befér a memóriába, nagyon gyors lekérdezéseket futtathatunk.
  - Korlátok: írási műveleteknél a diszk szűk keresztmetszet
- Master-slave replikáció
  - Működés: a MySQL-hez hasonlóan. Beállítható, hogy egyes slave-ek ne léphessenek elő masterré (read-scaling).
  - Korlátok: a replikáció szükségessége miatt minden értéket minden szerveren tárolunk
- Alkalmazásszintű sharding
  - Működés: alkalmazásszinten döntjük el, hogy mely csomópontok mely szerverekre kerüljenek.
  - Korlátok: domain-specifikus tudást igényel, hogy az egyes csomópontokat értelmes módon szétosszuk. Ennek implementációja általában nem triviális.

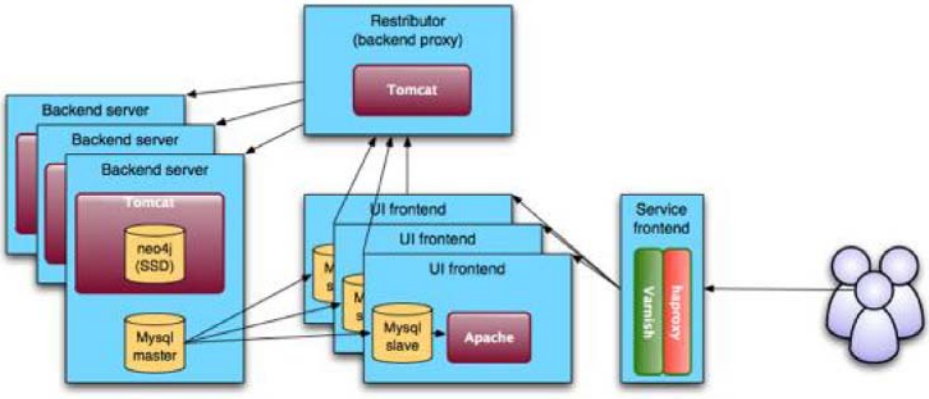
Egy magyarországi alkalmazás: SZTAKI szótár

A népszerű SZTAKI szótár 2012-ben teljes megújuláson esett át (12: ábra, 13. ábra), ennek során a felhasználó felületen túl a szótár adatbázisrétege is átalakult.



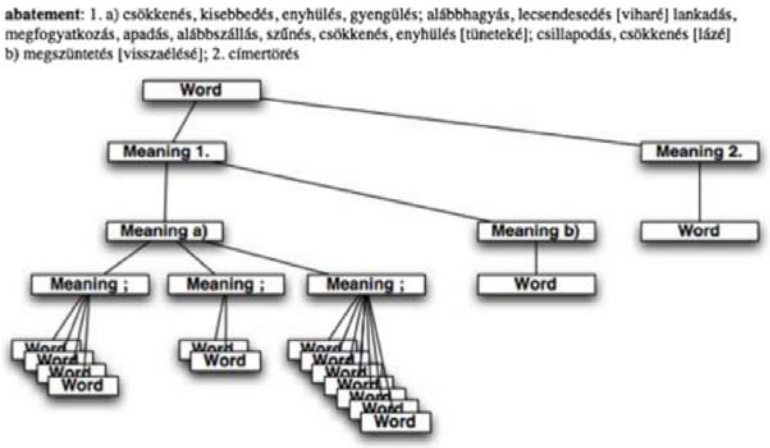
12: ábra: Régi szótár

13. ábra: Az új SZTAKI szótár (2012. novemberi állapot)



14. ábra: SZTAKI szótár architektúra [15]

A SZTAKI szótár gráfadatbázisában a szótár szavai a gráfadatbázis csomópontjaiként tárolódnak, amelyeket gazdag kapcsolatrendszer köt össze. Ez lehetővé teszi, hogy az egyes nyelvekhez tartozó, eltérő jelzeteket kényelmesen, a gráf részeként kezeljék.
A gráfalapú tárolás jó minőségű keresztszótárak létrehozását is lehetővé teszi, pl. egy magyar–angol és egy angol–német szótárból készíthető magyar–német szótár.



15. ábra. Szócikk ábrázolása [15]

Függelék

A rabbithole eszközről

A http://console.neo4j.org/ oldalon a *rabbithole* nevű eszköz fut. Ez egy olyan REPL konzol (Read, Evaluate, Print, Loop), amelyen a felhasználók kényelmesen, regisztráció és telepítés nélkül próbálhatják ki a Neo4j gráfadatbázis lekérdezőnyelvét.
Arra az esetre, ha az oldal az előadás ideje alatt nem lenne elérhető, előkészítettem egy virtuális gépet, amelyen a rabbithole eszköz fut. Az alábbiakban dokumentálom az eszköz fordítását.

A rabbithole eszköz fordítása

A leírás Ubuntu Linux 12.10-hez készült, de más Linux operációs rendszereken is alkalmazható. Windows operációs rendszer esetén a szükséges eszközöket nem a csomagkezelő segítségével, hanem a megfelelő weboldalakról letöltve kell telepíteni.
Szükséges alkalmazások:

1. Maven (`maven` csomag) a függőségek feloldásához.
2. JRE (pl. az `openjdk-6-jre` csomag) a futtatáshoz.
3. git kliens a forráskód letöltéséhez (`git` csomag). Helyettesíthető a `wget` eszközzel vagy böngészőből letöltéssel.
4. Eclipse IDE for Java Developers fejlesztőkörnyezet a fordításhoz (letölthető a http://www.eclipse.org/downloads/ címről).

Az 1–3. alkalmazások az alábbi parancsokkal telepíthetők.

```
sudo apt-get install maven
sudo apt-get install openjdk-6-jre
sudo apt-get install git
git clone https://github.com/neo4j-contrib/rabbithole.git
```

A forráskódból az alábbi paranccsal generálhatunk Eclipse projektet:

```
mvn eclipse:eclipse
```

A függőségek letöltése több percig is eltarthat. Ezután az Eclipse-ben a File | Import | General | Existing Projects into Workspace segítségével importálhatjuk a projektet.
Sajnos a 2012. november 3-án letöltött verzió az alábbi hibát adja fordítás során a `CypherQueryExecutor.java` 29. sorára:

```
The constructor ExecutionEngine(GraphDatabaseService) is undefined
CypherQueryExecutor.java
```

A hibás sor:

```
executionEngine = new org.neo4j.cypher.ExecutionEngine(gdb);
```

Javítása: vegyük fel az importok közé az alábbi sort.

```
import org.neo4j.kernel.logging.BufferingLogger;
```

Módosítsuk a hibás sort az alábbira.

```
executionEngine = new org.neo4j.cypher.ExecutionEngine(gdb, new
BufferingLogger());
```

Ezután a File | Export | Java | Runnable JAR file | Extract requires libraries into generated JAR | OK parancssorozattal exportálhatunk futtatható fájlt. A kapott `rabbithole.jar` fájlt a rabbithole könyvtárba, az src könyvtár mellé kell tenni. Az alkalmazás az alábbi paranccsal indítható:

```
java -jar rabbithole.jar
```

Irodalomjegyzék

[1] Gajdos Sándor, *Adatbázisok,* 4. kiadás, 2012.

[2] Pramod J. Sadalage, Martin Fowler, *NoSQL Distilled – A Brief Guide to the Emerging World of Polyglot Persistence,* Addison-Wesley, 2012.

[3] Barabás Ákos, Szárnyas Gábor, Gajdos Sándor, *NoSQL adatbázis-kezelők,* 2012.

[4] *Cypher – A view from a recovering SQL DBA,* http://systay.github.com/blog/2011/11/06/cypher---a-view-from-a-recovering-sql-dba/

[5] Bin Shao, Haixun Wang, Yatao Li, *The Trinity Graph Engine,* Microsoft Research whitepaper, 2011., http://research.microsoft.com/pubs/161291/Trinity.pdf

[6] Marko A. Rodriguez, Peter Neubauer, *Constructions from Dots and Lines,* Bulletin of the American Society for Information Science and Technology, 2009., http://arxiv.org/abs/1006.2361

[7] *Neo4j dokumentáció,* http://docs.neo4j.org/

[8] *Neo4j – The Benefits of Graph Databases,* OSCON 2009, http://www.slideshare.net/emileifrem/neo4j-the-benefits-of-graph-databases-oscon-2009

[9] *NoSQL Frankfurt 2010 – The GraphDB Landscape and sones,* http://www.slideshare.net/ahzf/nosql-frankfurt-2010-the-graphdb-landscape-and-sones

[10] *Neo raises $10.6M for Neo4j as graph DBs take off,* http://gigaom.com/cloud/neo-raises-10-6m-for-neo4j-as-graph-dbs-take-off/

[11] *Graph startup Neo raises $11M as specialized databases take hold,* http://gigaom.com/data/graph-startup-neo-raises-11m-as-specialized-databases-take-hold/

[12] *Yet Another Validation for NoSQL World,* http://nosql.mypopescu.com/post/279181036/yet-another-validation-for-nosql-world

[13] *Tinkerpop – Property Graph Model,* https://github.com/tinkerpop/blueprints/wiki/Property-Graph-Model

[14] Jim Webber, *Scaling Neo4j with Cache Sharding and Neo4j HA,* http://jim.webber.name/2011/02/scaling-neo4j-with-cache-sharding-and-neo4j-ha/

[15] Pataki Balázs, *SZTAKI Szótár – Olyan jó, hogy nem találom a szavakat,* http://nws.niif.hu/ncd2012/docs/ehu/022.pdf

[16] Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski, *Pregel: a system for large-scale graph processing,* http://dl.acm.org/citation.cfm?id=1807184

[17] *The igraph library,* http://igraph.sourceforge.net/

[18] Marko Rodriguez, *Titan: The Rise of Big Graph Data,* 2012, http://www.slideshare.net/slidarko/titan-the-rise-of-big-graph-data