Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Bokányi Balázs

A LEGKITERJEDTEBB, KÖZÖSSÉGILEG SZERKESZTETT ÁLTALÁNOS TUDÁSBÁZIS TAXONÓMIAI ELEMZÉSE

KONZULENS

Simon Gábor

BUDAPEST, 2020

Tartalomjegyzék

Összefoglaló

Napjaink legkiterjedtebb, közösségileg szerkesztett enciklopédikus tudásbázisa a Wikidata. Az emberi beavatkozások mellett számtalan szoftveres automatizmus aktívan bővíti és módosítja, nem csoda hát, hogy jelenleg is több tízmillió entitásról tartalmaz tényállításokat. A tudásbázison belül az egyes példányok egy típusrendszerhez kapcsolódnak, mely típusrendszer elemei is hasonlóan dinamikusan szerkeszthetők. A különféle módosítási folyamatok eredményeképpen egy igen komplex, sajátos, tipikus eszközökkel nehezen átlátható és vizsgálható séma jött létre.

Jelen munkában bemutatjuk a Wikidata típusrendszer felépítését, jellegzetességeit. Az általános tulajdonságokon túlmenően a vizsgálatok alapvetően a séma problémáinak feltárására irányulnak. Probléma alatt nemcsak a hagyományos értelemben vett modellezési hibákat értjük, mint például köröket a típusok gráfjában, hanem a tudásbázissal dolgozó szerkesztők, felhasználók munkáját megnehezítő sajátosságokat is.

Általános tudásbázisoknál, azaz nem szakértői rendszereknél ugyanis jelentős probléma, hogy a laikus felhasználók mentális sémája és a tudásbázis valódi sémája között jelentős eltérés lehet. A mentális séma általában jóval egyszerűbb, mint a számos folyamat által hosszabb időn keresztül csiszolt tudásbázisséma. Mindez megannyi felhasználási esetben a rendszer és használója közötti meg nem értéshez, fennakadásokhoz vezethet. Kutatásunk kiterjed ezen jelenséghez kapcsolható sémajellemzőkre is.

Abstract

Wikidata is the most extensive, community-edited encyclopedic knowledge base today. In addition to human interactions, countless software automations are actively expanding and modifying it, so it's no wonder that it contains factual statements about tens of millions of entities. Within the knowledge base, each instance is linked to a type system, the elements of which can be similarly dynamically edited. As a result of the various modification processes, a very complex scheme has been created, which is challenging to understand and examine using typical tools.

In this paper, we delve into the structure and characteristics of the Wikidata type system. Besides general properties, our research is aimed at exploring the problems of the scheme. We consider not only traditional modeling errors as an issue, such as circles in the type graph, but also features that make the job of editors and users working with the knowledge base cumbersome.

It is a common case with general-purpose knowledge bases, i.e. systems aimed at non-experts that there is a significant difference between the mental scheme of lay users and the actual scheme of the knowledge base. The mental schema is usually much simpler than the knowledge base schema polished over a long period of time by many processes. All this can lead to misunderstandings between the user and the system in many use-cases. Our research also examines schema characteristics related to this phenomenon.

1 Bevezetés

Napjainkban az internet elérhetőségének és kiterjedtségének köszönhetően szinte bármilyen kérdésünkre percek alatt választ kaphatunk. Olyan, nagy mennyiségű információ és tudás gyűlt össze weblapok sokaságán, ami évtizedekkel ezelőtt elképzelhetetlen lett volna. Az egyes keresőmotorok feltérképezik és nyilvántartják a világháló legkülönbözőbb pontjaiban elérhető oldalakat, majd a felhasználók keresőkifejezése alapján prezentálják a relevánsakat.

Ez azonban csak egy módja az információ nyilvánossá és hozzáférhetővé tételének. Manapság számtalan tudásbázis létezik és növekszik napról napra, óráról órára. Közös jellemzőjük, hogy az adatokat szimbolikus módon, tipikusan tények és azokból építkező szabályok formájában tárolják, melyeket aztán egy tudás-alapú rendszer vagy következtetőgép felhasználhat. Legfőbb céljuk, hogy az információt gyorsan és a lehető legpontosabban elérhetővé tegyék. A legtöbb szakértői és döntéstámogató szoftver mögött van egy ilyen speciális adatbázis.

Ugyanakkor ismeretbázis és ismeretbázis között számottevő különbségek lehetnek. Az adatbázis technikai megvalósítása, az adatok tárolásának és elérhetőségének milyensége mellett megkülönböztetünk általános és speciális célú tudásbázisokat. Utóbbiak célközönsége egy adott tématerülethez tartozó személyek összessége, pl. biológusok (rendszertan), jogi képviselők stb. Adataik szigorúan az adott tudományághoz kapcsolódnak, arról tartalmaznak minél pontosabb, naprakészebb, teljesebb tudást. Bővítésüket és karbantartásukat a szakterülethez tartozó emberek végzik, hiszen ők lesznek a felhasználóik is.

Ezzel szemben az általános tudásbázisokban mindenről található információ. Legyenek azok híres vagy kevésbé híres emberek, különböző állatfajok, járművek, publikációk, tárgyak, filmek, sportágak vagy országok, egytől egyig szerepelnek vagy szerepelhetnek az adatbázisban. A speciális célú adatbázisokkal ellentétben, a szerkesztők is a mindennapi felhasználók, tartalmilag pedig egy minél szerteágazóbb, sokszínűbb információhalmaz létrehozása és bővítése a cél. Az eltérő tartalmi célok miatt az adatokkal kapcsolatban is más tulajdonságok lesznek fontosak. Teljességre törekedni az adatbázis egészét nézve nem lehet, csak azon belül egy-egy területen, pl.

focisták vagy autóversenyzők összegyűjtésében. Az egyes állítások igazsága természetesen itt is lényeges szempont.

Manapság a legnagyobb, általános célú tudásbázisok a YAGO [10], a DBPedia [14] és a Wikidata [12]. A YAGO (Yet Another Great Ontology) nyílt forráskódú tudásbázis több, mint 10 millió entitásról tartalmaz 100 milliónál is több állítást. A benne található információt három különböző forrásból nyerik ki, melyek a Wikipedia, a WordNet és a GeoNames. Aktívan fejlesztik és karban tartják, legújabb verziója, a YAGO 4 [15] 2020 márciusában jelent meg.

A DBpedia célja a Wikipediában található információ kinyerése és strukturált formában elérhetővé tétele. A felhasználók számára lehetővé teszi a Wikipedia erőforrások kapcsolatainak és tulajdonságainak lekérdezését, beleértve más, kapcsolódó adatbázisokra való hivatkozásokat is.

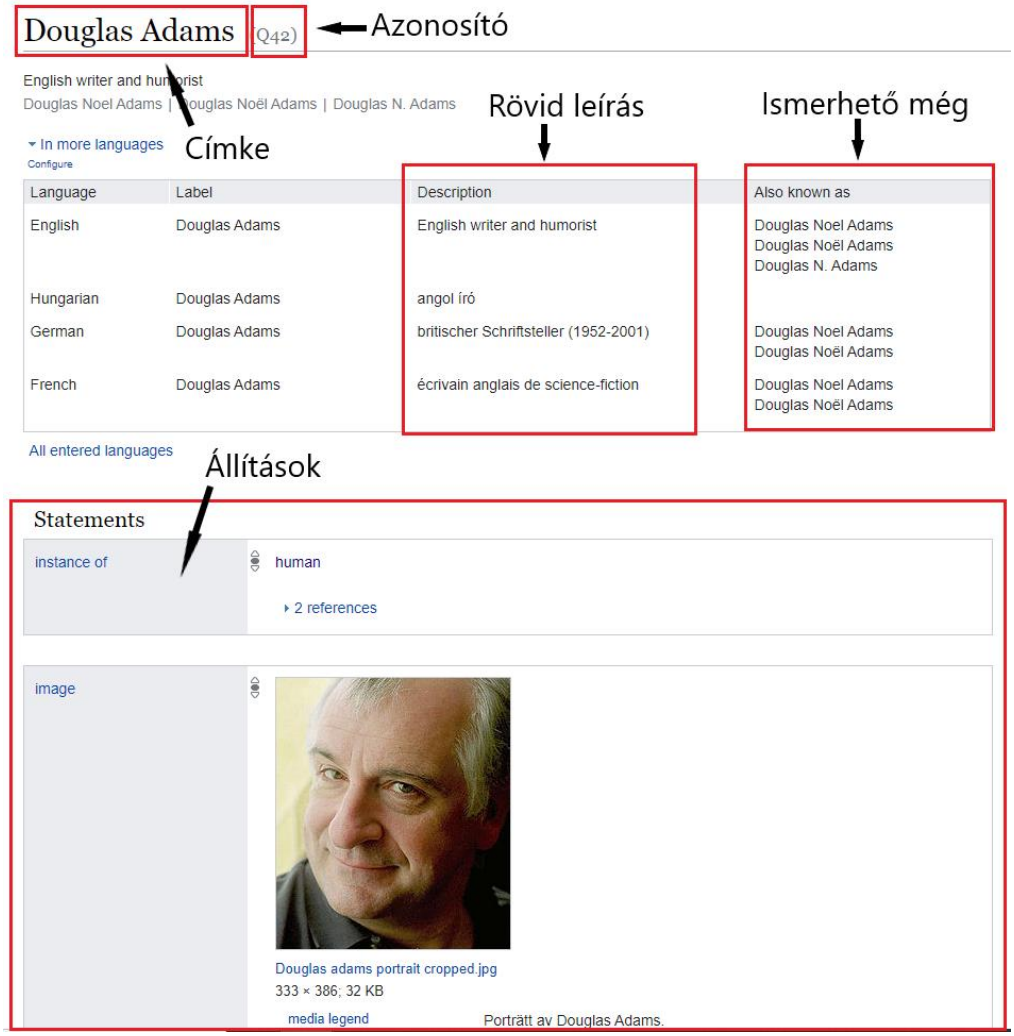
A három közül a legkiterjedtebb és legtöbb felhasználóval rendelkező tudásbázis a Wikidata. A 2012-ben indult Wikimedia projekt az erős közösségi támogatásnak köszönhetően gyorsan növekedett és egyre népszerűbb lett. Az általános célú ismeretbázisok közt egyeduralkodóvá válásában nagy szerepet játszott az is, hogy 2014-ben a Google megszüntette a Freebase [16] projektjét, és felajánlotta az addig összegyűjtött adatait a Wikidatának. Mára a Wikidata más Wikimedia projektek, köztük a Wikipedia központi adattárházául szolgál. Több, mint 50 millió entitásról tárol adatokat 700 milliónál is több állítás formájában.

A heti rendszerességgel előállított adatfájlok JSON, RDF vagy XML[1] formátumban tartalmazzák a teljes adatbázist. Mi a 2019. szeptember 16-án elérhetővé tett JSON verzión végeztük a lekérdezéseket. A fájlok mellett lehetőség van webes felületen keresztül[2] is vizsgálni a típusokat és tulajdonságokat.

Minden entitáshoz tartozik (többek közt) egy azonosító, címke, rövid leírás, álnevek és az állítások. Ezek közül a vizsgálatok során az azonosítót és az állításokat használtuk, a címkék csak megkönnyítik az entitások beazonosítását.

[1] https://www.wikidata.org/wiki/Wikidata:Database_download

[2] https://www.wikidata.org/wiki/Wikidata:Main_Page



1.1. ábra Egy entitás oldala a webes felületen

A lényegi információt tehát az állítások kódolják. Ezek az állítások a JSON fájlban úgynevezett *Snackeken*[3] belül érhetők el. Ahhoz, hogy alany – állítmány – tárgy hármasokkal tudjunk dolgozni, előbb ki kellett őket nyerni a forrásfájlból.

A Wikidatában nincsenek automatikusan ellenőrzött kényszerek állítások felvitelére. A nagy számú szerkesztők a legjobb tudásuk szerint, lényegében bármilyen tulajdonságot megadhatnak egy-egy típushoz. Rajtuk kívül automatizmusok is bővítik az adathalmazt. A sok szerkesztési mód és laikus felhasználó miatt az adatok minősége nem egységes, számos esetben találhatunk hibás, vagy hiányos entitásokat.

Emiatt nagyon fontos, hogy legyenek olyan elemzések, melyek feltárják a súlyosabb hibákat, anomáliákat, hiszen az adatminőség magas szinten tartása

[3] https://www.mediawiki.org/wiki/Wikibase/DataModel/JSON

mindenkinek az érdeke. Jelen munkánk célja, hogy a tudásbázis feltérképezésével betekintést nyújtsunk a valós felépítésbe, összevessük az elméleti működést a gyakorlati megvalósítással és ezáltal rávilágítsunk a problémás területekre.

A következő szakaszban ismertetjük a Wikidatával kapcsolatos főbb kutatási területeket, majd részletezzük az elvégzett elemzéseket. Minden lekérdezéshez a magyarázatra szoruló kódrészlet megtalálható a szövegben, míg a teljes lekérdezések a függelékben.

2 Irodalmi áttekintés

A Wikidata, mint legnagyobb, közösségileg szerkesztett tudásbázis számos tudományos munka tárgyát képezi. Jelen fejezetben ismertetem azokat a fő irányokat, melyekben a kutatások zajlanak. Nagy népszerűségnek örvend a Wikidata teljességének vizsgálata, melynek során automatizált módszerekkel próbálják azokat a típusokat vagy egyedeket kiszűrni, melyek bővítésre, további kitöltésre szorulnak. Hasonlóan gyakoriak azok a tanulmányok, amik vandalizmus detektálással, azaz szándékosan hibás adatok felvitelének megtalálásával foglalkoznak. Ezek mellett készülnek a legnagyobb tudásbázisok adatminőségét összehasonlító vagy kimondottan a Wikidata adatminőségét elemző dolgozatok is.

2.1 Teljesség vizsgálata

A tudásbázisok teljessége olyan, az adatok minőségére utaló mérőszám, melynek segítségével elképzelésünk lehet arról, hogy a szükséges információ milyen mértékben van jelen az adott adathalmazban. Ezek a vizsgálatok hasznosak az olyan osztályok, típusok azonosításában, melyek erősen hiányosak, kitöltésre, bővítésre szorulnak. A tudásbázis-független megközelítések előnye, hogy több információhalmazon alkalmazhatók, hátrányuk viszont, hogy egy bizonyos adatbázison rosszabb eredményeket adhatnak. A szabályokon alapuló módszerek [1] erősen építenek az adatok szemantikai jelentésére, például, ha a szülei tulajdonságra két tárgya van az adott alanynak, akkor azt teljesnek mondják.

Ennél automatikusabb vizsgálat az, ha a tudásbázisban megtalálható, ’hasonló’ példányok tulajdonságait vetjük össze [2]. A hasonlósági tényező lehet az embereknél pl. a foglalkozás. Ebben az esetben van egy kiválasztott alany, akihez a kontroll csoportot a hasonló foglalkozású emberek jelentik. Megkeresik azt a néhány leggyakoribb tulajdonságot, ami a csoportban előfordul, a vizsgált személynek viszont nincs. A tulajdonságok kontroll csoportban való előfordulási gyakoriságát is figyelembe véve kialakul egy mérőszám, ami jó indikátora a vizsgált alany kitöltöttségi szintjének.

Nemcsak az egyes entitások kitöltöttsége, hanem egy adott típushoz tartozó példányok számossága is vizsgálható statisztikai módszerekkel [3]. Ennek segítségével megválaszolhatók olyan kérdések, mint pl. „Tartalmazza a tudásbázis a Föld összes

vulkánját?” A választ a példányokra irányuló módosítások gyakoriságából adják meg, amit pedig a szerkesztési előzményeken keresztül érnek el.

Jelen munkánk nem kísérel meg ehhez hasonló, teljességre vonatkozó állításokat tenni. Más megközelítéssel, de az egyik célunk nekünk is az, hogy olyan rávilágítsunk olyan entitásokra, entitás-csoportokra, melyek bizonyos szempontból hiányosak és kiegészítésre szorulnak.

2.2 Vandalizmus detektálás

Az erős közösségi támogatásnak köszönhetően gyors ütemben növekszik az adathalmaz mérete, sok száz felhasználó hozzájárulásával óráról órára bővülnek az állítások. A módosítások nagyobb része jó szándékú, új és pontos információ felvitele a cél. Egy kis részük azonban kifejezetten káros, hiszen tudatosan hibás vagy értelmetlen adatokkal akarják bővíteni a tudásbázist. A vandalizmus áldozatául eső leggyakoribb típusok például az emberek adatlapjai, különösen a magas rangú politikusoké [5].

A szándékosan romboló hatású módosításokat gépi tanuláson alapuló módszerek jól ki tudják szűrni. Egy lehetséges megközelítés, ha minden szerkesztésből bizonyos jellemzőket nyerünk ki, melyekkel aztán egy pl. döntési fát betanítva eldönthető, hogy a vizsgált változtatás vandalizmus-e [4]. A módszerek teljesítményének ellenőrzéséhez rendelkezésre áll a Wikidata Vandalism Corpus[4], ami felcímkézett adatokat tartalmaz. Ez az a megközelítés, amit a Wikidata jelenleg is használ a vandalizmus kiszűrésére.

Ennek az eszköznek egy hátránya, hogy nagy mértékben diszkriminálja az új és az anonim felhasználókat. A továbbfejlesztett változatában [7] ezt a problémát sikerült kiküszöbölni, ám a módszer pontossága kis mértékben csökkent.

Van olyan törekvés is, ami a manuális ellenőrzésre beküldött módosítások számát igyekszik csökkenteni úgy, hogy az nem megy a pontosság rovására [6]. A gépi tanuláson alapuló módszer használatával nagy mértékben redukálható a jó szándékú szerkesztések felesleges ellenőrzése.

[4] https://zenodo.org/record/3250651

2.3 Adatminőség

A manapság elérhető, legnagyobb általános tudásbázisok adatminőségének összehasonlítása népszerű kutatási terület. A DBpedia és a Wikidata [8] összevetéséhez számos különböző szempontot megvizsgálnak, úgymint az adatok valósághoz való viszonyát, teljességét, tárolási módját vagy elérhetőségét. Ehhez hasonló az a tanulmány is [9], melyben az említett két tudásbázis mellett a YAGO jellemzőit is vizsgálják.

Ugyancsak az adatok minőségét és teljességét hivatott mérni az a módszer, mely során a Wikidatában található adatokat a Wikipedia szöveges oldalaiból kinyert állításokkal hasonlítják össze [11]. A vizsgálat célja a Wikipedia oldal alapján annak eldöntése, hogy egy adott személynek hány gyermeke van. A kapott eredményt a Wikidata állítással összehasonlítva (ha egyáltalán van ilyen) ellenőrizhető az állítás helyessége, illetve ha nincs erre vonatkozó állítás, akkor automatikusan bővíthető a szóban forgó személy adatlapja.

3 Saját megoldás ismertetése

A Wikidata adathalmaz komplexitása és nagysága (~700 GB) az elemzések futtatásához megköveteli egy elosztott, jól skálázódó, alfeladatokat párhuzamosan futtató rendszer használatát. Több százmillió állítás között keresni, millió soros táblákat illeszteni, több százezer egyedi azonosítón csoportosítani; ezek mind-mind olyan tipikus műveletek, melyeket egyrészt szinte minden lekérdezéshez el kell végezni, másrészt hagyományos eszközökkel csak jelentős időbefektetés mellett lehet megoldani (ha egyáltalán lehetséges).

Már a jelen tanulmány kezdetén is egyértelmű volt, hogy saját számítógép kevéssé alkalmas a feladatra, valamilyen felhőszolgáltatás felé kell fordulnunk. Olyan platformra volt szükségünk, ami lehetővé teszi ilyen méretű adatok tárolását, illetve biztosít egy olyan analitikai szoftvert, amivel az elemzések percek, rosszabb esetben néhány óra alatt elkészülnek.

A választott felület végül a Microsoft Azure lett, melynek több szolgáltatását is igénybe vettük, mire ’elemzésre kész’ állapotba kerültek az adatok. A lekérdezéseket Data Lake Analytics[5] használatával futtattuk, aminek az egyik legnagyobb előnye a nagyfokú párhuzamosíthatóság. Lekérdező nyelve az U-SQL[6], amiben deklaratív módon adhatjuk meg, hogy mi is az, amire kíváncsiak vagyunk. Az imperatív nyelvekkel szemben alacsonyabb kifejező erővel rendelkezik, ám a végrehajtást így nem nekünk kell definiálni, rábízhatjuk azt az adatbázisra. Ennek eredménye egy nagymértékben párhuzamos futtatásra optimalizált lekérdezés lesz, ami gyorsabb futtatási időt eredményez egy imperatív módon megfogalmazott lekérdezésnél.

Az analitikai infrastruktúra inicializálása után kezdődhetett az érdemi munka. A Wikidata tudásbázis két olyan, kitüntetett szereppel rendelkező tulajdonságot definiál, melyek a taxonómia elengedhetetlen részét képezik. Az *instance of*[7] kapcsolat azt adja meg, hogy egy adott entitás példánya egy osztálynak (pl. Douglas Adams az ember

[5] https://docs.microsoft.com/en-us/azure/data-lake-analytics/

[6] https://docs.microsoft.com/en-us/azure/data-lake-analytics/data-lake-analytics-u-sql-get-started

[7] https://www.wikidata.org/wiki/Property:P31

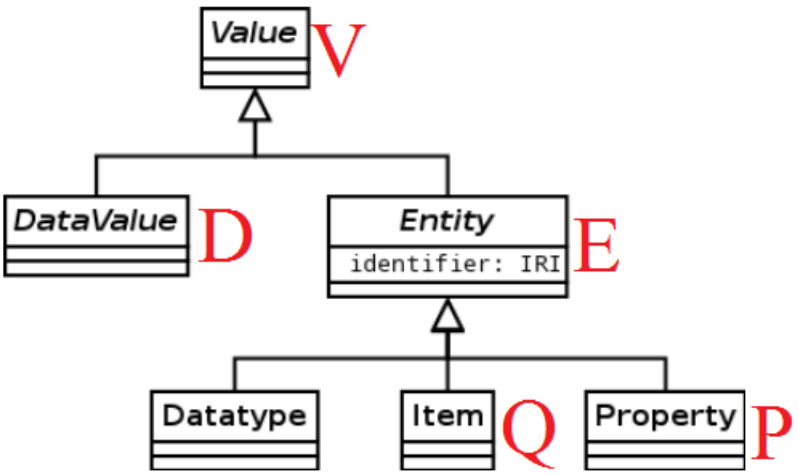
típus egy példánya), míg a *subclass of*[8] tulajdonság a típusrendszer leszármazási hierarchiáját építi fel[9].

Ahhoz, hogy elemzéseket készíthessünk, először szükségünk volt egy fogalmi rendszerre, melyben bizonyos tulajdonságokkal rendelkező entitásokat definiálunk. A definíciók középpontjában a fent említett két tulajdonság áll, azok megléte vagy épp hiánya számottevő különbségeket eredményez az entitásokra nézve.

A kétértelműség elkerülése és a precíz definiálás érdekében nemcsak folyószöveggel, hanem formálisan is megadtuk az egyes definíciókat, lekérdezéseket. A következő szakaszban ezt a formális megfogalmazást ismertetjük, mielőtt sor kerülne a lekérdezések és eredmények részletezésére.

3.1 Formalizmus

A Wikidata tudásbázis állítások (Statements, S) definiálásával reprezentálja a tárolt információt. Az állítások tulajdonságok (Properties, P) használatával épülnek fel, és entitásokat (Entities, E) kapcsolnak össze valamilyen értékkel (Values, V). Entitás lehet adatelem (Item, I) vagy más tulajdonság is, míg érték a konkrét tulajdonságtól függően, lehet adatelem, tulajdonság, valamilyen adattípus (Data Value, D), pl. URI, koordináta, szöveges adat, vagy a két speciális, ismeretlen (*some value*) és hiányzó érték (*no value*).



3.1. ábra Wikidata adatmodell

[8] https://www.wikidata.org/wiki/Property:P279

[9] https://www.mediawiki.org/wiki/Wikibase/DataModel

A tudásbázis tehát nem más, mint állítások összessége; ezen állításokból kiindulva fogalmazták meg [13] az alábbi definíciót a Wikidatára nézve:

Definíció: Legyen Q a Wikidata adatok halmaza, P a Wikidata tulajdonságok halmaza, és jelölje D az összes lehetséges adattípust. Jelölje ekkor E := Q U P az entitások halmazát, és V := D U E az értékeket. Ekkor a tulajdonságokon keresztül megfogalmazott állítások olyan W hozzárendelések, melyekre igaz, hogy W: P -> (E × V × (P × V)). Azaz, W minden tulajdonsághoz három tagból álló W(p) relációt rendel. A háromtagú reláció elemei az állítás tárgya (subject, s), értéke (value, v) és megszorításai (qualifiers, q). Egy tetszőleges állítást ekkor az <s,v,q> ϵ W(p) jelöl.

A fenti definíció egy több attribútumból álló relációs struktúraként adja meg az adathalmazt. Közérthető jelölése alkalmassá teszi arra, hogy ezt felhasználva egyszerűen, de nagy pontossággal tudjuk definiálni az általunk megalkotott fogalmakat.

3.2 Elnevezések és definíciók

Jelen szakaszban ismertetjük azokat a fogalmakat, melyekkel a bizonyos tulajdonságokkal rendelkező adatokat tudjuk megkülönböztetni. Az egyes definíciók kivétel nélkül a *subclass of* és *instance of* állításokra épülnek.

3.2.1 Hivatkozott (implicit) típus

Olyan adatelem, ami szerepel tárgyként *instance of* állításban.

HT := { ∀i | ∃s,q: <s, i, q> ϵ W(P31), s ϵ E, i ϵ Q, q ϵ (P×V)}

Hivatkozott típus lesz minden, olyan QXX azonosítóval rendelkező érték, aminek legalább egy példánya van.

3.2.2 Taxonómiai (explicit) típus

Olyan adatelem, aminek van *subclass of* állítása és az *Entity*.

TT := { ∀i | ∃v,q: <i, v, q> ϵ W(P279), i ϵ Q, v ϵ V, q ϵ (P×V)} U {Q35120}

A taxonómiai típusok alkotják a leszármazási hierarchiát a *subclass of* állításon keresztül. A gyökér elem, az *Entity* (Q35120) kivételével mind megtalálhatók a fentebb megadott módon. A legabsztraktabb típusnak nincsen *subclass of* állítása, mégis a taxonómiai típusokhoz tartozik, hiszen közvetlen vagy közvetett módon minden típus belőle származik. Fontos, hogy egy típusnak több közvetlen őse is lehet.

3.2.3 Valódi hivatkozott típus

Hivatkozott és taxonómiai típus is.

VT := { HT ∩ TT}

3.2.4 Példány

Olyan érték (*Value*, V), aminek van *instance of* állítása (szerepel alanyként), de nincs *subclass of* állítása.

PLD := { ∀i | ∃v,q: <i, v, q> ϵ W(P31) és !∃w,r: <i, w, r> ϵ W(P279), i ϵ Q, v,w ϵ V, q,r ϵ (P×V)}

A példányok lesznek az egyes típusok konkrét megtestesítői: egy adott ember (pl. Douglas Adams[10]), egy híres állat (pl. Koko[11]), egy konkrét publikáció (pl. The Extended Mind[12]) stb.

3.2.5 Egyéb

Eldönthetetlen; nincs se instance of, se subclass of állítása.

EB := { ∀i | !∃v,q: <i, v, q> ϵ W(P31) és !∃w,r: <i, w, r> ϵ W(P279), i ϵ Q, v,w ϵ V, q,r ϵ (P×V)}

Az egyéb típusba tartozó adatelemek számossága az adathalmaz minőségének egy jó mutatója. Elméletben azt várnánk, hogy egy entitás vagy a taxonómia felépítésében vesz részt, vagy pedig példányként van jelen a rendszerben. Ezzel szemben a gyakorlatban látni fogjuk, hogy számos olyan adatelem van, melyek egyik csoportba sem sorolhatók.

3.2.6 Levél

Taxonómiai típus, de nem szerepel *subclass of* állítás tárgyaként.

L := { ∀i | !∃s,q: <s, i, q> ϵ W(P279), i ϵ TT, s ϵ E, q ϵ (P×V)}

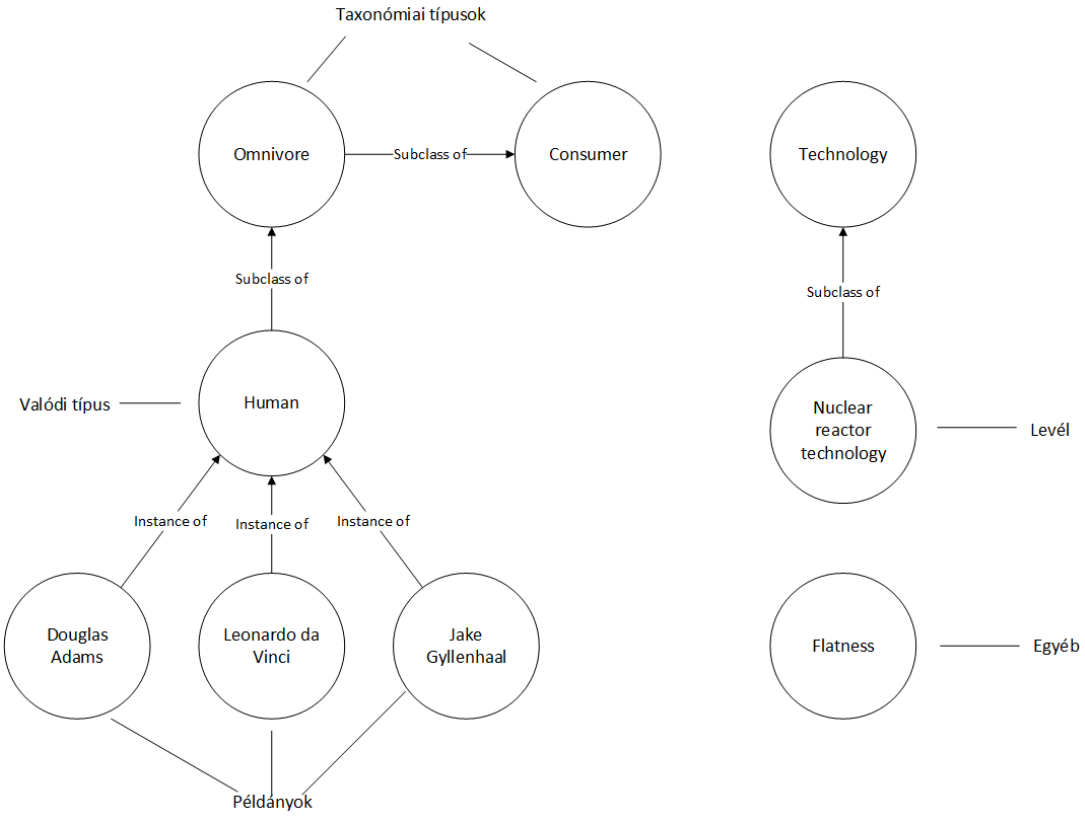
---

[10] https://www.wikidata.org/wiki/Q42

[11] https://www.wikidata.org/wiki/Q1348219

[12] https://www.wikidata.org/wiki/Q1362699

Levelek lesznek azok az explicit típusok (vagyis nekik még van *subclass of* állításuk), melyek a leszármazási hierarchia „szélén” helyezkednek el.



3.2. ábra Definíciók szemléltetése

3.3 Kitérő – U-SQL

Az U-SQL az Azure Data Lake Analytics deklaratív lekérdező nyelve. Egy tipikus U-SQL *script* három fő lépésből áll: az adatok beolvasása fájlból vagy táblából, adatok feldolgozása, majd fájlba / táblába írása. A feldolgozás során úgynevezett *rowset*-ek keletkeznek, melyek leginkább az adatbázis táblákhoz hasonlatosak. Minden lépésben egy vagy több *rowset* a bemenet, ahogy a kimenet is. Fontos, hogy minden közbülső lépésnek közvetlenül hozzá kell járulnia a végezetül pl. fájlba írt eredményhez.

Az alábbi példa az állítások felhasználásával megkeres öt embert az adatbázisban. Azt mondjuk, hogy ember lesz minden olyan alany, akinek van *instance of – human* állítása.

```
//bemeneti adatok felolvasasa
@data =
    SELECT * FROM SubjPropObj;
```

```
//minden alany, ahol instance of – human szerepel
@stmtFilter =
    SELECT DISTINCT [Subj]
    FROM @data
    WHERE Prop == "P31" AND Obj == "Q5"; //instanceOf

//random 5 kivalasztasa
@stmtSample=
    SELECT [Subj]
    FROM @stmtFilter
         SAMPLE ANY(5);

//fajlba iras
OUTPUT @stmtCount
TO "/wikidata/adla-teszt-lekerdezes.csv"
USING Outputters.Csv(outputHeader:true,quoting:true);
```

Említés szintjén volt arról szó, hogy a lekérdezések futtatásakor a forrás adatokat már nem a JSON fájlból nyerjük ki, hanem egy adatbázis táblából. A forrásfájl feldolgozása idő- és erőforrásigényes művelet, ezért célszerű azt egyszer elvégezni, majd a releváns információt egy gyorsan elérhető helyen tárolni.

Először egy olyan adatbázis táblát hoztunk létre, ami az *Id, Type, Labels, Descriptions, Aliases, Claimes, Sitelinks* oszlopokat tartalmazza. Minden sor egy entitáshoz tartozó információt kódol. Az *Id* mező az azonosítót, a *Type* a szóban forgó entitás típusát (*Item* vagy *Property*) jelzi. A többi oszlop a webes felületen is látott *Címke*, *Részletes leírás*, *Ismerhető még* és *Állítások* részeket tartalmazza byte tömbök formájában. Erre a táblára egyes lekérdezések során `@WikidataImports` néven hivatkozunk.

A második lépésben ezt az adatbázis táblát bontottuk tovább. A *Claims* oszlopban található byte-tömböt beolvastuk és a benne található *snackekből* megkapott állításokat egy három oszlopot tartalmazó U-SQL táblába mentettük el. Szintén, egy sor felel meg egy állításnak, a három oszlop pedig rendre az állítás alanya (*Subject*, *Subj*), tulajdonsága (*Property*, *Prop*) és tárgya (*Object*, *Obj*). A vizsgálatok során az U-SQL kódban erre a táblára `@data`-ként hivatkozunk.

3.4 Elemzések és eredmények

A fentebb megadott definíciók használatával ismertetjük a különböző elemzéseket, melyeket az adathalmazon végeztünk. A lekérdezéseket mind formális, mind informális módon megadjuk. Emellett minden vizsgálatnál szerepel az azt

megvalósító U-SQL lekérdezés releváns részlete (tehát pl. az adatok felolvasása vagy fájlba írás nem). A teljes lekérdezések megtalálhatók a függelékben.

3.4.1 Típusrendszerrel kapcsolatos adatok minősége, kitöltöttsége

Adathalmaz mérete. A legelső kérdés, ami felmerülhet minden, az adathalmazzal foglalkozó felhasználó fejében, hogy pontosan mekkora is a tudásbázis? Hány különböző adattal dolgozunk? 10 millió vagy esetleg 100 millió? Nagyon fontos, hogy mielőtt bármilyen elemzésnek nekilátnánk vagy következtetést levonnánk, tisztában legyünk a konkrét számokkal.

Az első lekérdezés célja tehát ennek meghatározása. Keressük az összes olyan adatelemet, ami egy állítás alanyaként vagy tárgyaként szerepel, majd vesszük a kapott halmaz számosságát.

Formálisan:

$SA := \{ \forall s \mid <s,o,q> \in W(p), s \in Q, o \in V, q \in (P \times V), p \in P\}$

$SO := \{ \forall o \mid <s,o,q> \in W(p), s \in E, o \in Q, q \in (P \times V), p \in P\}$

$Adatelem := \{(SA \cup SO) / (SA \cap SO)\}$

Fontos, hogy azokat az adatelemeket, melyek szerepelnek állítás alanyaként és tárgyaként is, ne számoljuk kétszer.

U-SQL lekérdezésként megfogalmazva:

```
@allDistSubject =
    SELECT DISTINCT Subj
    FROM @data;

@allDistObj =
    SELECT DISTINCT Obj AS Subj
    FROM @data;

@union =
    SELECT Subj FROM @allDistSubject
    UNION DISTINCT
    SELECT Subj FROM @allDistObj;
```

Az elemzéshez használt adathalmazban a különböző adatok száma **57.900.681.** Ekkora információ-mennyiséget automatikus feldolgozás nélkül már lehetetlen lenne átlátni, kezelni. Általánosságban mondhatjuk, hogy ha valamilyen közismert eseményről, személyről vagy dologról szeretnénk többet megtudni, akkor jó eséllyel találunk róla információt az adatbázisban.

Hiányos adatelemek. Felmerülhet ugyanakkor a kérdés, hogy miért nem elég csak az állítások alanyai közt keresni a különböző adatokat? Van olyan adat, amire létezik hivatkozás, de magáról az adatról semmilyen állítás nincsen? A válasz az, hogy igen, méghozzá nem is elhanyagolható mennyiségben. Ezeket az adatokat megkaphatjuk úgy, hogy az összes különböző, tárgyként szereplő alanyból elvesszük az alanyként szereplő adatokat.

Formálisan:

Hiányos adatelem := {SA / SO}

U-SQL lekérdezésként megfogalmazva:

```
@hianyos_adat =
    SELECT *
    FROM @allDistObj
    EXCEPT DISTINCT
    SELECT *
    FROM @allDistSubject;
```

A hiányos elemek száma **62.537,** pl. ilyen a *flatness*[13]. Habár elsőre nem feltétlenül egyértelmű, hogy ilyesfajta, hiányos adatok miért szerepelnek az adatbázisban, gondoljunk bele a következő szituációba. Adott egy felhasználó, aki fel akar venni egy új állítást valamilyen adathoz, és annak tárgyaként olyan értéket akar megadni, ami nem szerepel a tudásbázisban. Ekkor felveszi azt egy új értékként, kap egy azonosítót, majd hivatkozik rá az állításban. A felhasználót ekkor nem feltétlenül érdekli, hogy arról az új értékről lesz-e bármilyen állítás, ő csak hivatkozni akart rá, a kitöltését ráhagyja ’valaki másra’.

Ez egy teljesen életszerű eset, melynek eredményeként ilyen hiányos adatokat kapunk. Ezek megkeresése és tulajdonságokkal való felruházása jelentős mértékben hozzájárulna az adatbázis minőségének javításához.

Eldönthetetlen típusok. Tudjuk, hogy elméletben a tudásbázis egy típushierachiából és a hozzá kapcsolódó példányokból épül fel. A valóságban azonban, ahogy szinte minden elméleti megfontolásnál, ebben az esetben is vannak kivételek. A következő lépésként nézzük meg, hogy hány Egyéb típusú adat szerepel a Wikidatában, azaz nincs se *instance of*, se *subclass of* állításuk.

[13] https://www.wikidata.org/wiki/Q5457948

Ezen adatok halmazát megkaphatjuk például úgy, hogy megkeressük az összes *instance of* és az összes *subclass of* állítás nélküli adatot, majd vesszük a kapott két halmaz metszetét.

Formálisan:

Van_instance := { ∀s| ∃<s,o,q> ϵ W(P31), s ϵ Q, o ϵ V, q ϵ (P×V)}

Nincs_instance := {Adat / Van_instance}

Van_subclass := { ∀o| ∃<s,o,q> ϵ W(P279), s ϵ E, o ϵ Q, q ϵ (P×V)}

Nincs_subclass := {Adat / Van_subclass}

Egyéb := {Nincs_instance ∩ Nincs_subclass}

U-SQL lekérdezésként megfogalmazva:

```
@van_instanceof =
    SELECT DISTINCT Subj FROM @data
    WHERE Prop == "P31";

@nincs_instanceof =
    SELECT Subj FROM @osszesitem
    EXCEPT DISTINCT
    SELECT * FROM @van_instanceof;

@van_subclass =
    SELECT DISTINCT Subj
    FROM @data
    WHERE Prop == "P279";

@nincs_subclass =
    SELECT Subj FROM @osszesitem
    EXCEPT
    SELECT Subj FROM @van_subclass;

@egyiksem =
    SELECT * FROM @nincs_instanceof
    INTERSECT
    SELECT * FROM @nincs_subclass;
```

Az így kapott halmaz, vagyis az Egyéb típusba tartozó adatok számossága **1.053.877** darab. Ez az összes adathoz (~57 millió) képest egy jelentős mennyiség, főként, ha belegondolunk, hogy elméletben minden adatot egyértelműen vagy a taxonómiához, vagy a példányokhoz be tudunk sorolni. Ilyen például az óbudai Fogadalmi oltár[14], amit ki lehetne egészíteni az *'instance of – sculpture*[15]' állítással.

[14] https://www.wikidata.org/wiki/Q1002383

Taxonómiáról leszakadt adatelemek. Az egyértelmű megkülönböztethetőség hiányánál súlyosabb hiba, ha bizonyos részek le vannak szakadva a taxonómiai fáról. Ez a gyakorlatban úgy nyilvánul meg, hogy vannak olyan adatok, melyekre mutat *subclass of* állítás, ám nekik nincsen *subclass of* állításuk. A legegyszerűbb esetben hozzá lehetne kötni ezeket a részeket a taxonómia gyökeréhez, az *Entity*-hez, de némi kutatás után minden bizonnyal lehetne találni kevésbé absztrakt típust is, amihez kapcsolódhatnak.

Formálisan:

Elszigetelt adatelem := { ∀o| ∃<s,o,q> ϵ W(P279), s ϵ E, o ϵ Nincs_subclass, q ϵ (P×V)}

U-SQL lekérdezésként:

```
@filter =
    SELECT DISTINCT Obj FROM @data AS d
          INNER JOIN (SELECT Subj FROM @nincs_subclass) AS n
                        ON d.Obj == n.Subj
    WHERE d.Prop == "P279";
```

Ahogy a formális megadásban, úgy a konkrét lekérdezésben is felhasználtuk a korábban előállított Nincs_subclass adathalmazt. Halmazműveletek helyett egy belső illesztéssel könnyen megkapjuk a keresett adatokat.

A lekérdezés eredményeként megkapjuk, hogy **9582** elszigetelt adatelem van a tudásbázisban, amik nem kapcsolódnak a taxonómiához. Egy ilyen részfa gyökere például a Földköpeny[16], amihez *subclass of* állítással kapcsolódik a Mezoszféra[17]. Ezen a példán láthatjuk azt is, hogy a mi definícióink szerint a Földköpeny Példány, hiszen van *instance of* állítása, de nincs *subclass of* állítása. Ebben az esetben tehát egy Példányhoz kapcsolódik a Mezoszféra *subclass of* állítással, ami a Földköpeny absztrakt típusosságát sugallja. Mivel nincs semmilyen kényszer vagy korlátozás a Wikidata szerkesztésére és a típusok megadására vonatkozólag, ilyen és ehhez hasonló ellentmondások előfordulhatnak és elő is fordulnak. Ezek feloldásához nem feltétlenül

---

[15] https://www.wikidata.org/wiki/Q860861

[16] https://www.wikidata.org/wiki/Q101949

[17] https://www.wikidata.org/wiki/Q257408

elég egy laikus szerkesztő tudása, hanem valamilyen, az adott szakterületen jártas felhasználó véleményére is szükség van.

Árva adatelemek. Eddig a pontig csak olyan adatokat vizsgáltunk, melyekről vagy tudunk valamit, vagy egy másik adat hivatkozik rájuk, vagyis, az állításokon keresztül legalább egy másik entitáshoz kapcsolódnak. Található azonban a típusrendszerben **2,199,183** darab olyan adat is, ami teljesen elszigetelt: nincs rájuk hivatkozás, és nekik sincs állításuk. Többségükben ezek teljesen kitöltetlen oldalak (pl. Q1001504[18]), a kivételeknél pedig csak címke vagy egy rövid leírás van megadva, pl. dark-energy star[19].

Formálisan:

Árva_adat := { ∀i| !∃<i,o,q> ϵ W(p) és !∃<s,i,q> ϵ W(p), s,i ϵ Q, o ϵ V, q ϵ (P×V)}

U-SQL lekérdezésként megfogalmazva:

```
@RawData =
    SELECT [Id]
    FROM WikidataImport
    WHERE Type == "item";

@tablaban =
    SELECT DISTINCT Subj FROM SubjPropObj
    UNION
    SELECT DISTINCT Obj FROM SubjPropObj;

@except =
    SELECT * FROM @RawData
    EXCEPT
    SELECT * FROM @tablaban;
```

Ahhoz, hogy ezeket megtaláljuk, ki kellett nyerni a forrás *JSON* fájlból minden olyan azonosítót, aminek a típusa *'Item'*, majd elvenni belőlük azokat, amik szerepelnek állításokban.

Ezek, az adatbázisban 'lebegő' csomópontok nagyon kevés, a legtöbb esetben semmilyen hozzáadott értéket nem tartalmaznak. Abban az esetben, ha csak az azonosító lett lefoglalva és nincs hozzá semmilyen címke vagy leírás, akár törölni is lehetne őket a tudásbázisból. Habár az egy időben létezhető azonosítók száma nem

[18] https://www.wikidata.org/wiki/Q1001504

[19] www.wikidata.org/wiki/Q1002787

korlátos, tehát ebből a szempontból lényegében mindegy, hogy szerepelnek-e vagy sem, feleslegesen növelik a forrás fájl méretét, és a tudásbázis általános minőségén is csak rontanak. Ha valamilyen szöveges tulajdonság már meg lett adva egy azonosítóhoz, érdemes lenne azokat tovább bővíteni, kitölteni, és a taxonómiához hozzákapcsolni.

3.4.2 Taxonómiai és hivatkozott típusok

Az előző szakaszban ismertettük a tudásbázisban megtalálható adatelemek számosságát, tulajdonságait, hiányosságait. Tudjuk, hogy az adatbázis egy leszármazási hierarchiából és a hozzá kapcsolódó példányokból épül fel. Arról azonban nincsen még információnk, hogy a ~57 millió adat milyen arányban oszlik el a taxonómiát alkotó csomópontok és a példányok között. Jelen fejezet az explicit és az implicit típusok fogalmának segítségével adja meg erre, és ehhez hasonló kérdésekre a választ.

Taxonómiai típusok. Elsőként nézzük meg közelebbről a taxonómiai típusokat. A 3.2.2-es fejezet definíciója szerint minden olyan adat, ami szerepel alanyként *subclass of* állításban, taxonómiai típus lesz. Erre lekérdezés nélkül is könnyedén tudunk példát mutatni, hiszen pl. az ember[20], a húsevők[21], vagy az elektronikus dokumentum[22] is ide tartozik. Nem kell tehát mást tenni, mint megkeresni minden *subclass of* állítást, és venni az összes, egyedi alanyként szereplő adatelemet.

Formálisan:

Tax_típus := { ∀i | ∃v,q: <i, v, q> ϵ W(P279), i ϵ Q, v ϵ V, q ϵ (P×V)}

U-SQL lekérdezés:

```
@taxtipus =
    SELECT DISTINCT Subj
    FROM @data
    WHERE Prop == "P279";
```

Ezzel az egyszerű lekérdezéssel egy kivételével megtalálható mind a **2.433.345** taxonómiai típus. Volt már arról szó korábban, hogy a típusrendszer gyökerét alkotó *Entity* nem rendelkezik *subclass of* állítással, ám funkcióját tekintve egyértelműen a

[20] https://www.wikidata.org/wiki/Q5

[21] https://www.wikidata.org/wiki/Q81875

[22] https://www.wikidata.org/wiki/Q694975

taxonómiai típusokhoz sorolandó. Ezt a lekérdezésben nem, az elemzésekben viszont annál inkább figyelembe kell venni.

Hivatkozott típusok. ~2,5 millió adat alkotja tehát a leszármazási hierarchiát. Hogy ez sok vagy kevés, az Olvasó megítélésére bízzuk; egy ekkora adathalmazban mindenesetre reális. Ennél is érdekesebb kérdés a hivatkozott típusok számossága. Az már ebből az eredményből is sejthető, hogy a tudásbázis nagy részét a példányok alkotják, de vajon hány különböző típus van, melyekhez tartozik legalább egy példány?

Formálisan:

HT := { ∀i | ∃s,q: <s, i, q> ϵ W(P31), s ϵ E, i ϵ Q, q ϵ (P×V)}, ami egyben a hivatkozott típusok definíciója is.

U-SQL lekérdezésként megfogalmazva:

```
@hivatkozott =
    SELECT DISTINCT Obj
    FROM @data
    WHERE Prop == "P31";
```

Ez az eredmény már sokkal meglepőbb, hiszen csak **62.775** típus van, ami rendelkezik példánnyal. Mielőtt bármire is következtetnénk ebből az értékből, érdemes megvizsgálni azt is, hogy pontosan hány példány szerepel a rendszerben. Az eddig elmondottak alapján, a taxonómiai és hivatkozott típusok elvételével maradó ~55 millió adat legnagyobb részének példánynak kell lennie.

Példányok számossága. Hasonlóan a hivatkozott típushoz, a Példány definíciója egyben formális lekérdezést is ad:

PLD := { ∀i | ∃v,q: <i, v, q> ϵ W(P31) és !∃w,r: <i, w, r> ϵ W(P279), i ϵ Q, v,w ϵ V, q,r ϵ (P×V)}

U-SQL lekérdezés:

```
@alanyok =
    SELECT DISTINCT Subj FROM @subjPropObj
    WHERE Prop == "P31";

@van =
    SELECT DISTINCT Subj FROM @subjPropObj
    WHERE Prop == "P279";

@peldanyok =
    SELECT * FROM @alanyok
    EXCEPT
    SELECT * FROM @van;
```

Az eredmény a várakozásokkal összhangban van, ugyanis **54.413.460** példány, vagyis konkrét személy[23], film[24], tudományos munka[25], könyv[26], vagy akár egy bolygó[27] található a tudásbázisban.

Mit is jelent ez pontosan? Az ismeretbázis ~54 millió példánya, vagyis 94 százaléka csupán ~60 ezer típushoz, az összes adat egy tized százalékához kapcsolódik. Ez a jelenség több dolog együttes fennállásával magyarázható. Először is, az emberek általában valamilyen létező vagy létrehozott, élő vagy élettelen, de konkrét személyről, tárgyról, eseményről szoktak beszélgetni vagy az iránt érdeklődni. Ezeket a konkrét dolgokat rendszerbe foglaló taxonómia viszont tudományos munka eredménye, nem egy hétköznapi beszédtéma. Mivel a Wikidatát többségében laikus felhasználók szerkesztik, akik a saját érdeklődési körükbe tartozó személyekről, tárgyakról rendelkeznek elegendő ismerettel ahhoz, hogy egy Wikidata oldalt feltöltsenek releváns tartalommal, egyenesen adódik, hogy a Példányok túlnyomó többségben lesznek az adatbázisban.

A hivatkozott típusok száma részben ugyancsak az emberi viselkedéssel, közös jellemzőkkel magyarázható. Általánosságban kijelenthetjük, hogy az emberek nagy többségét ugyanazok a dolgok érdeklik: filmek, könyvek, folyóiratok, más emberek, műsorok, együttesek, zenészek stb. Következésképp vannak népszerű típusok, amiknek egyre több példányuk lesz, és vannak kevésbé népszerűek, melyek esetlegesen nem is szerepelnek a tudásbázisban. Másrészt pedig, ha nem a teljes adathalmazhoz viszonyítjuk a számosságukat, hanem reálisan belegondolunk, a ~60 ezer különböző típus egyáltalán nem kevés. Legtöbbünk nagyjából százat, míg a legjobbak is csak néhány százat tudnának felsorolni némi gondolkodás után. Jó példa ez arra, hogy minden adatot a megfelelő kontextusban, kellő körültekintéssel szabad csak értelmezni.

Valódi hivatkozott típusok. Eddig külön-külön beszéltünk taxonómiai és hivatkozott típusokról, de ahogy más esetekben, úgy itt sem tiltja semmilyen kényszer,

---

[23] https://www.wikidata.org/wiki/Q172724

[24] https://www.wikidata.org/wiki/Q18002795

[25] https://www.wikidata.org/wiki/Q13442814

[26] https://www.wikidata.org/wiki/Q929184

[27] https://www.wikidata.org/wiki/Q308

hogy egy adott adat ne tartozhasson mindkét csoportba. Sőt, ebben az esetben az a helyes, ha egy hivatkozott típus egyben taxonómiai típus is. Hiszen ez azt jelenti, hogy van legalább egy absztrakt felmenője, és megvan rá az esély vagy definíció szerint ténylegesen kapcsolódik a leszármazási hierarchiához. Nevezzük ezeket a típusokat Valódi hivatkozott típusoknak, és határozzuk meg az ide tartozó adatelemek számát.

Formálisan:

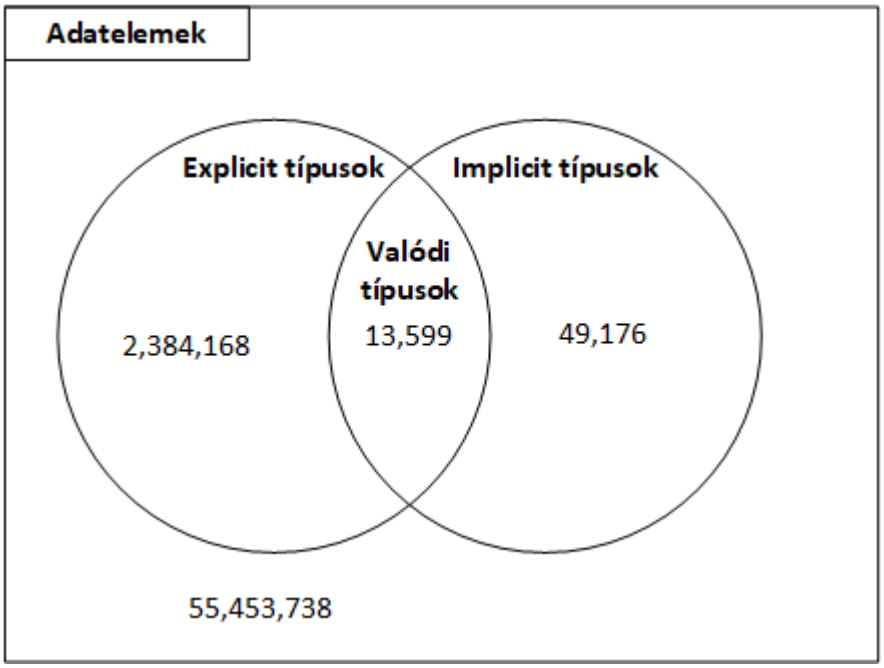
VT := { HT ∩ TT}

U-SQL lekérdezésként:

```
@taxEsHiv =
    SELECT * FROM @taxtipus
    INTERSECT DISTINCT
    SELECT * FROM @hivatkozott;
```

Ahogy a tényleges lekérdezésben, úgy a formális megadásban is felhasználtuk a korábban definiált fogalmakat.

Az összes hivatkozott típus 78%-a, **49.176** egyben taxonómiai típus is. Figyelembe véve, hogy elméletben mindegyiknek ide kellene tartoznia, ~80%-os helyesség elég kevés. Ez főként akkor probléma, ha van egy olyan hivatkozott típus, ami sok példánnyal rendelkezik, ám le vannak szakadva a központi leszármazási fáról. A tudásbázis minőségén jelentősen javítana, ha ezek a típusok megfelelően lennének csatlakoztatva a taxonómiához.



3.3. ábra Adatelemek megoszlása a típusok között

Azok az adatelemek, amik a taxonómiai típusokhoz tartoznak, de nem valódi típusok, alkotják az absztrakt típusok csoportját. Ezek nem hibásak vagy hiányosak, a leszármazási hierarchia szerves részét képezik. Az objektum orientált fogalmi rendszerben ekvivalensek az absztrakt osztályokkal.

Formálisan:

Absztrakt típus := { TT \ HT}

Valódi típusok példányai. A többség azonban rendelkezik *subclass of* állítással, így jó eséllyel közvetlenül vagy közvetetten, de rendelkeznek a közös ős osztállyal, az *Entity*-vel. Említés szintjén volt már arról szó, hogy a rendszerben lévő példányok nagy száma az emberi viselkedést tükrözi, hiszen javarészt felhasználók bővítik az adathalmazt. Azt viszont még nem tudjuk, hogy ezek a példányok hogyan oszlanak el a valódi típusok között? Egyenletesen, vagy esetleg vannak központi szereplők, melyek a példányok nagy részével rendelkeznek? Melyik típusnak lehet a legtöbb példánya? Ezekre a kérdésekre keressük most a választ.

Formálisan:

VT_állítások := { $\forall <s,o,q> \in W(31) \mid s \in PLD, o \in VT, q,r \in (P \times V)$}

U-SQL lekérdezésként:

```
//megkeressuk azokat az instance of allitasokat, melyek targya valodi
tipus
@valodira_mutato_allitasok =
    SELECT s.Subj, s.Prop, s.Obj FROM @subjPropObj AS s
                INNER JOIN (SELECT Obj FROM @realTypes) AS r
                         ON s.Obj == r.Obj
    WHERE s.Prop == "P31";

//aztan meghagyjuk azokat, ahol az alanynak peldany
@filtered =
    SELECT v.Subj, v.Prop, v.Obj FROM @valodira_mutato_allitasok AS v
                  INNER JOIN (SELECT Subj FROM @nincs_subclass) AS n
                  ON v.Subj == n.Subj;


//vegul csoportositunk a valodi tipusok szerint, kulon oszlopba felveve az
elfordulasok, azaz a peldanyok szamat
@countOccurances =
    SELECT Obj, COUNT() AS Occurance, 1 AS Same
    FROM @filtered
         GROUP BY Obj;
```

Az így létrehozott adatbázis-tábla olyan formában tartalmazza a minket érdeklő információt, melyből könnyedén lehet statisztikai mutatókat (példányok átlagos száma, szórása, minimuma, maximuma, mediánja) számolni.

```
@jellemzok =
    SELECT MAX(Occurance) AS Maximum, MIN(Occurance) AS Minimum,
AVG(Occurance) AS Average, STDEV(Occurance) AS Spread
    FROM @countOccurances;

@median =
    SELECT DISTINCT Same, PERCENTILE_DISC(0.5) WITHIN GROUP(ORDER BY
Occurance) OVER(PARTITION BY Same) AS Median
    FROM @countOccurances;
```

A kapott eredmények ugyancsak meglepőek. Egy valódi típusnak átlagosan **1171** példánya van, **109,800**-as szórás értékkel. A minimum példányszám **1**, a maximum **22,641,730**, a középérték pedig **4**. Ebből egyértelműen látszik, hogy van néhány olyan típus, ami a példányok túlnyomó többségével rendelkezik, míg az összes többi típusnak néhány példánnyal kell beérnie.

Ez a jelenség, a középpontok létezése egyáltalán nem egyedi. Az egyik legszemléletesebb példa a skálafüggetlen hálózatok felépítése, ahol néhány csomópont rendelkezik a kapcsolatok többségével. Ezt Barabási Albert-László Behálózva [17] című könyvében (ötödik és hatodik fejezet) részletesen ismerteti.

További érdekesség, hogy a legtöbb példánnyal rendelkező típus a tudományos munka[28]. Ezek az adatelemek a teljes adatbázis közel 40 százalékát teszik ki. Csak találgatni tudunk, hogy miért szerepelnek ilyen hangsúlyosan, véleményünk szerint elképzelhető, hogy egy meglévő, akadémiai tanulmányokat tartalmazó adatbázist beolvasztottak a Wikidatába.

3.4.3 Taxonómiai típusok hálózata

A Wikidata típusrendszerét megvizsgáltuk már azokból a szempontokból, hogy hány különböző adatelem alkotja, vannak-e olyan részek, melyek nem kapcsolódnak a leszármazási hierarchiához, vagy melyik csomópont helyezkedik el a fa gyökerében. A most következő fejezetben a taxonómiát vesszük górcső alá, meghatározzuk többek között a kiinduló pontból elérhető útvonalakat, azok tulajdonságait (átlagos, maximális és minimális mélységüket).

[28] www.wikidata.org/wiki/Q13442814

Elméletben a taxonómiát egy olyan gráffal lehetne reprezentálni, melyben a gyökér elem az *Entity*, a csomópontok az egyes taxonómiai típusok, egy A és egy B csomópont között pedig akkor van A -> B irányított él, ha van olyan állítás, hogy B – P279 – A. Fontos, hogy egy típusnak több ősosztálya is lehet. A statisztikai mutatók számításához előbb fel kell térképezni az összes, gyökér elemből levél csomópontig vezető útvonalat.

Ehhez az *Entity*-ből kiinduló, rekurzív szélességi keresést végeztünk a *subclass of* kapcsolatok mentén. Olyan x0, x1, x2, …, xn sorozatokat kerestünk, melyekre igaz, hogy a kiinduló elem, x0 = *Entity* (Q35120), és minden két, egymást követő x_i-re van olyan állítás, hogy x_i – P279 – x_{i+1}.

Formálisan:

Útvonalak := { $x_0,x_1,x_2,\ldots,x_n$ | x_0 = Q35120, $\exists<x_i, x_{i+1}, q> \in$ W(P279), $x_0,x_1,x_2,\ldots,x_n \in$ TT, i = 0,1,…,n-1}

U-SQL lekérdezésként megfogalmazva:

```
CREATE FUNCTION dbo.SimpleRecursion(@path TABLE(), @founded TABLE(),
@pathLen int = 0, @MaxIterations int = 5, @data TABLE())
RETURNS @result TABLE()
AS
BEGIN



    @left_join =
        SELECT f.Subj AS LastElement, p.Paths.Contains(f.Subj) AS
CircleFlag, new SqlArray<string>(p.Paths){f.Subj} AS Paths
               FROM (SELECT * FROM @path WHERE CircleFlag == false AND
LastElement != null) AS p
                    LEFT OUTER JOIN (SELECT Subj, Obj FROM @data) AS f
                                    ON f.Obj == p.LastElement;

    @results =
        SELECT * FROM @founded
        UNION ALL
        SELECT * FROM @path WHERE LastElement == null;

    IF (@pathLen >= @MaxIterations) THEN
        @result = SELECT * FROM @results
        UNION ALL
        SELECT * FROM @left_join;
    ELSE
        @result = dbo.SimpleRecursion(@left_join, @results, @pathLen + 1,
@MaxIterations, @data);
    END;

END;
```

A lekérdezés menete a következő. Ahogy imperatív nyelvekben, úgy itt is definiálni kell egy függvényt, ami hívni fogja önmagát. Paraméterként megkapja az adott iterációban részben már felfedezett útvonalakat (`@path`), a már megtalált és véges útvonalakat (`@founded`), a *subclass of* állításokat tartalmazó táblát (`@data`), a jelenlegi iteráció számát (`@pathLen`) és a maximum iterációk számát (`@MaxIterations`). Az útvonalak bővítéséért egy külső illesztés felel, ami az adott pontig felfedezett útvonal-részletek utolsó eleme alapján keres folytatást a `@data` táblában. Ha van ilyen, akkor a következő csomópontot hozzáfűzzük az eddig megtalált útvonalhoz, ha nincs, akkor null érték jelzi az út végét.

Az illesztés után eltároljuk a megtalált útvonalakat, ha nem értük el a maximális iteráció számot, akkor a megfelelő paraméterekkel meghívjuk újra a függvényt, ellenkező esetben végére ér a hívás és visszatérünk az összes megtalált útvonallal.

Az első teszt-lekérdezések után hamar nyilvánvalóvá vált, hogy az elméleti feltételezések, mint más esetekben is, a gyakorlatban itt is megdőltek. A taxonómia ugyanis számos kört tartalmaz, melyeknek jelenlétét a lekérdezéskor is kezelni kellett. A végtelen futás elkerülése érdekében bevezettük a *CircleFlag* változót, ami egy adott útvonalra nézve azt mondja meg, hogy a legutoljára hozzá vett csomópont szerepelt-e már korábban, valamelyik lépésben. Ha igen, azokat az útvonalak egyszerűen figyelmen kívül hagyjuk és nem bővítjük tovább.

Az így megtalált útvonalakat egy listában eltárolva már könnyedén tudtuk elemezni. Az adatbázis-táblában lévő sorok száma megadja az összes elérhető útvonalat, míg a listák hossza az útvonalak hosszát. Ezek alapján, a gyökér csomópontból **80.219.024** különböző módon juthatunk el levél-csomópontig, az útvonalak átlagos hossza **14**, szórása **4,2**. A legrövidebb **3** taxonómiai típust tartalmaz, míg a leghosszabb **42**-t.

Ahhoz, hogy jobban el tudjuk képzelni a típusgráf alakját, meghatároztuk az előbb említett levél csomópontok számát. Definíciónk szerint, *Levél* lesz minden olyan taxonómiai típus, ami nem szerepel *subclass of* állítás tárgyaként.

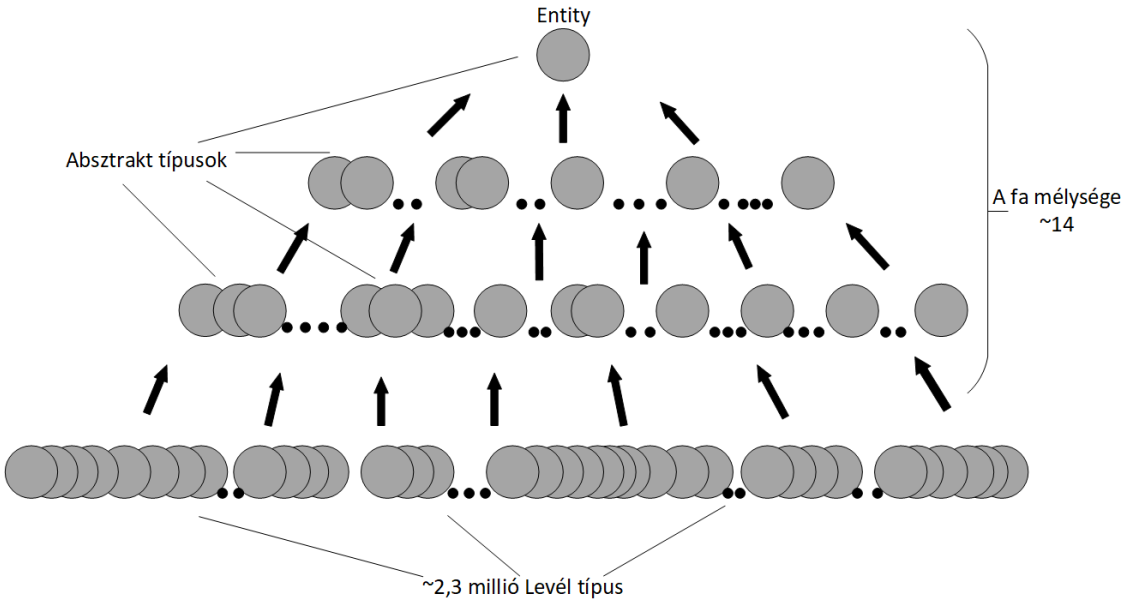
Formálisan:

Levél := { $\forall i \mid !\exists s,q: \langle s, i, q\rangle \in W(P279), i \in TT, s \in Q, q \in (P \times V)\}$

U-SQL lekérdezésként megfogalmazva:

```
@taxonomiai =
    SELECT DISTINCT Subj FROM @data
    WHERE Prop == "P279";

@szerepel =
    SELECT DISTINCT Obj FROM @data
    WHERE Prop == "P279";

@itemek =
    SELECT DISTINCT Subj FROM @data
    UNION
    SELECT DISTINCT Obj AS Subj FROM @data;

@nem_szerepel =
    SELECT * FROM @itemek
    EXCEPT
    SELECT * FROM @szerepel;

@levelek =
    SELECT *
    FROM @taxonomiai
    INTERSECT
    SELECT *
    FROM @nem_szerepel;
```

A ~2,4 millió csomópont közötti ~80 millió útvonal nagy mértékű összekapcsoltságot jelez. Levelekből **2,334,616** darabot találunk a típusrendszerben, ami az összes taxonómiai típus 96%-a. Láthatjuk továbbá azt is, hogy átlagosan 14 absztrakt típuson keresztül juthatunk el levél csomópontokig. Mindezek alapján egy nagyon széles, rengeteg végponttal rendelkező gráf alakja tárul elénk, melyben egy-egy absztrakt, nem-levél csomópont számos útvonalon megtalálható. Ezen forma kialakulásának oka lehet az, hogy egy új, leendő taxonómiai típus felvételekor a felhasználó csak egy megfelelően absztrakt típust keres, amihez tudja majd csatlakoztatni. A valóságban könnyedén előfordulhat, hogy az adott téma szakértője 3-4 köztes csomópontot is tudna javasolni, melyek az új típus és a kiszemelt ős közé illeszkednének, ám ezek felvételére (feltéve, hogy még nem szerepelnek) nem kerül sor.



3.4. ábra Taxonómia hozzávetőleges alakja

A taxonómiában található körök a technikai kihívásokon túl értelmezési nehézségeket is nyújtanak. Ez ugyanis azt jelenti, hogy egy adott típus absztrakt ősosztálya saját magának. Más, szigorúan szabályozott rendszerekben, pl. vírusok rendszertanában[29] ez nem fordulhat elő, és az általános gondolkodásnak is ellentmond. Példának okáért, nézzük meg a gráf[30] és a multigráf[31] kapcsolatát. Mindkettő taxonómiai típus, és mindketten a másik őseként szerepelnek.

Keletkezésük, mint más esetekben is, a laikus emberi interakciókra vezethető vissza. Egy adott típus taxonómiába ’bekötésénél’ nem várható el az, hogy minden, kapcsolódó útvonalat egyesével ellenőrizzen a felhasználó. Ezek kialakulását csak úgy lehetne elkerülni, ha az összes új, *subclass of* állítás felvételénél a tudásbázis automatikusan ellenőrizné, nem alakul-e ki kör a rendszerben. Ez azonban ellentmondana a Wikidata nyílt szerkesztési elveinek, így maradnak a típusrendszer körei a tudásbázis velejárója.

[29] https://talk.ictvonline.org/taxonomy/

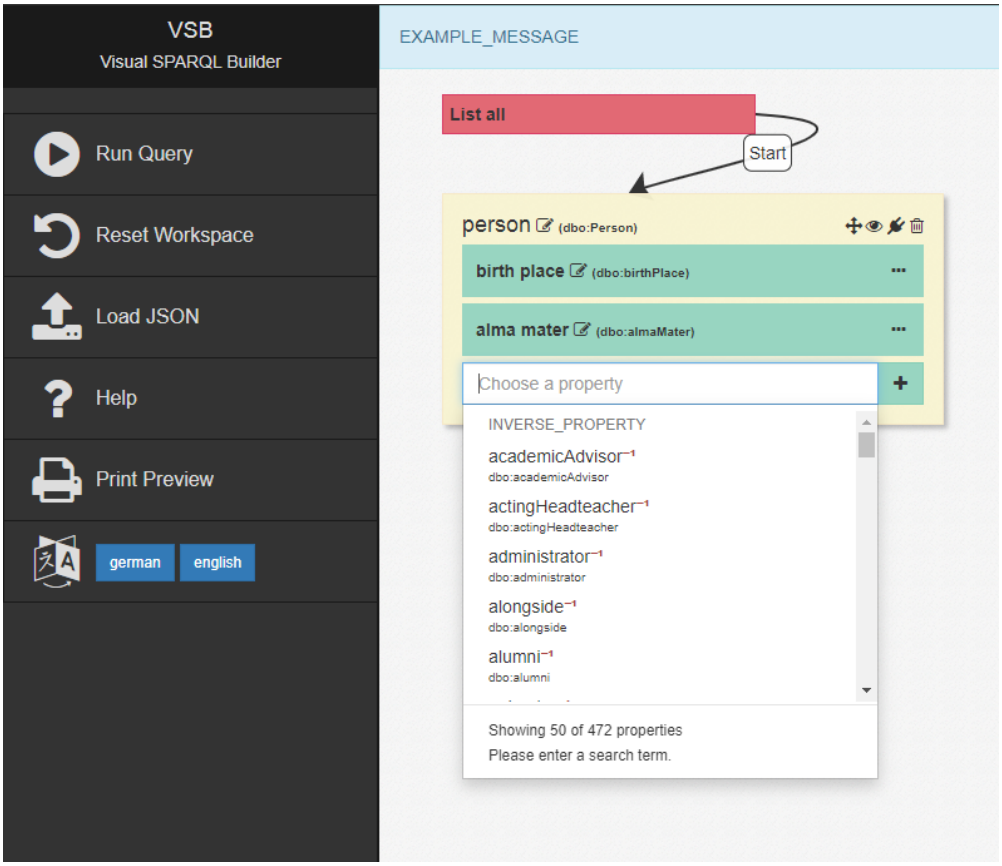
[30] https://www.wikidata.org/wiki/Q141488

[31] https://www.wikidata.org/wiki/Q2642629

3.4.4 Taxonómiai típusok tulajdonságai

Jelen munkánkhoz tartozó minden lekérdezést az Azure Data Lake Analytics használatával futtattunk. Az ADLA lekérdező nyelve, az U-SQL egy jól használható, funkciókban gazdag eszköznek bizonyult. Nem mindenki rendelkezik azonban SQL ismeretekkel azok közül, akik interakcióba lépnek vagy szeretnének lépni a Wikidatához hasonlóan nagy méretű tudásbázissal. Az ő dolgukat könnyítendő, léteznek vizuális lekérdezés-építő eszközök[32] is.

Ebben a szakaszban ismertetett elemzést ezek az alkalmazások inspirálták. Míg egy lekérdező nyelvnél a nyelvi sajátosságok elsajátítása jelenti a nehézséget, addig a vizuális esetben, ha kellően nagy a vizsgált adathalmaz, könnyedén elveszhetünk a típusok és tulajdonságok tengerében. Arra keressük a választ, hogy egy típus kiválasztásánál átlagosan hány különböző, megadható tulajdonság fog megjelenni, mennyire lesz nehéz a felhasználóknak azok közt eligazodni.



3.5. ábra Vizuális lekérdezés építése 472 különböző tulajdonságból

[32] https://github.com/leipert/vsb

Két irányból közelíthető meg ez a probléma: a Wikidata séma szerinti tulajdonságok felől, és a példányadatok felől. Utóbbi esetben meg kell keresni az adott típushoz tartozó összes példányt, és venni kell a példányok tulajdonságainak unióját. Ezek mellett vannak még az ún. inverz tulajdonságok, amit legegyszerűbb egy példán szemléltetni. Legyen a vizsgált típus az Ember (Q5), egy példánya Douglas Adams (Q42). Ha van olyan állítás, hogy Galaxis útikalauz (QXX) – írója (PYY) – Douglas Adams (Q42), aki egyébként Instance of (P31) – Human (Q5), akkor PYY egy megtalálandó inverz tulajdonság.

Séma szerinti vizsgálat esetén, a típusoknak lehet egy *'Properties for this type'* tulajdonsága, ami több, másik tulajdonságra mutat. Az inverz tulajdonságok az egyes tulajdonsághoz tartozó állításként jelennek meg: ha egy PYY tulajdonság *'Value type constraint'*-je QXX-re mutat, akkor PYY QXX egy inverz tulajdonsága.

A tulajdonságszámok meghatározását a taxonómiai típusokra korlátozva végezzük. A leszármazási hierarchia ismeretében tudjuk azt is, hogy a *subclass of* tulajdonságon keresztül megjelenő szülő csomópontok, mint ősosztályok viselkednek. Az objektum orientált világban megszokott leszármazással analóg módon azt mondjuk, hogy minden gyermek csomópont rendelkezik az összes, közvetlen vagy közvetett szülő csomópontjának tulajdonságaival is. Az átlagos tulajdonságszámok meghatározásánál tehát azokat is figyelembe kell venni.

3.4.4.1 Példányadatok szerinti vizsgálat

Mielőtt a szülőkkel is foglalkozhatnánk, meg kell mondani minden típushoz a tulajdonság számát. Ez a már említett módon, két lépésben történik, a normál és az inverz tulajdonságok összegzésével.

Formálisan:

Tulajdonság_pld_normál := { ∀p | ∃s,o,q: <s, o, q> ϵ W(p), s ϵ PLD, o ϵ TT, q ϵ (P×V)}

U-SQL lekérdezés:

```
//taxonomiai tipusok
@taxTypes =
    SELECT DISTINCT
        Subj
    FROM @subjPropObj
    WHERE Prop == "P279";
```

```
//instance of sorok
@instanceOfRows =
    SELECT * FROM @subjPropObj
    WHERE Prop == "P31";

//tax tipusok peldanyai
//peldany: van instance of de nincs subclass of allitasa
//elso lepes: valami-instance of-taxonomiai tipus
@filter =
    SELECT s.Subj AS Instance, s.Obj AS Types
    FROM @instanceOfRows AS s
         INNER JOIN
         (
         SELECT Subj
         FROM @taxTypes
               ) AS t
         ON s.Obj == t.Subj;

//masodik lepes: nincs subclass-instance of-taxonomiai tipus
//ehhez kell egy tabla, amiben azokat tarolon, akiknek nincsen subclass
of-ja
//legegyszerubb: osszes - van
@nincs_subclass =
    SELECT DISTINCT Subj FROM @subjPropObj
    EXCEPT
    SELECT * FROM @taxTypes;

@instancesOfTaxTypes =
    SELECT f.Instance, f.Types FROM @filter AS f
           INNER JOIN (SELECT Subj FROM @nincs_subclass) AS n
                    ON f.Instance == n.Subj;

//ezen peldanyok propertijei
@propsOfInstances =
    SELECT s.Subj, s.Prop
    FROM @subjPropObj AS s
         INNER JOIN (SELECT Instance FROM @instancesOfTaxTypes) AS i
         ON s.Subj == i.Instance;
```

A fenti lépéseket végrehajtva kialakul egy két oszlopot tartalmazó tábla, melyben a taxonómiai típusok példányai és a hozzájuk tartozó tulajdonságok szerepelnek. Ha az inverz tulajdonságokat is ilyen alakra hoznánk, egy unióval és a példányokon való csoportosítással megkapnánk a teljes tulajdonság listát.

Formálisan: a taxonómiai típusok példányainak megadása után (T_PLD) az inverz tulajdonságok halmaza könnyen kifejezhető.

T_PLD := { $\forall$i | $\exists$v,q: <i, v, q> $\in$ W(P31) és !$\exists$w,r: <i, w, r> $\in$ W(P279), i $\in$ Q, v $\in$ TT, w $\in$ V, q,r $\in$ (P×V)}

Tulajdonság_pld_inverz := { $\forall$p | $\exists$s,o,q: <s, o, q> $\in$ W(p), s $\in$ Q, o $\in$ T_PLD, q $\in$ (P×V)}

U-SQL lekérdezésként megfogalmazva:

```
//ami itt Obj, nekem Subj-kent kell az uniohoz
//az elozovel azonos formara hozom
@inverzPropOfInstances =
    SELECT s.Obj AS Subj, s.Prop
    FROM @subjPropObj AS s
         INNER JOIN (SELECT Instance FROM @instancesOfTaxTypes) AS i
                    ON s.Obj == i.Instance;
```

Ekkor már csak az említett unió műveletre és a csoportosításra van szükség.

```
@unionProperties =
    SELECT * FROM @propsOfInstances
    UNION
    SELECT * FROM @inverzPropOfInstances;

//taxonomiai tipusonkent csinalok egy group-by-t
//a hozzajuk tartozo property-k egy sqlArray-ben
@output =
    SELECT s.Types AS Obj, ARRAY_AGG(DISTINCT p.Prop) AS Prop
    FROM @unionProperties AS p
        INNER JOIN (SELECT Instance, Types FROM @instancesOfTaxTypes) AS s
           ON s.Instance == p.Subj
    GROUP BY s.Types;
```

Készen van tehát egy olyan tábla, melyben Példány – Tulajdonság lista alakban szerepelnek az adatok. A szülő csomópontok meghatározásához vissza kell nyúlni egy korábbi elemzéshez, melynek során eltároltuk a taxonómiában szereplő összes útvonalat.

Elsőként ki kell nyerni minden útvonalból a benne található típusokat úgy, hogy közben az az információ is megmarad, hogy melyik típust melyik útvonalból kaptuk. Ezt aztán tudjuk szűrni aszerint, hogy csak azokat a sorokat hagyjuk meg, ahol a típus olyan típus, aminek az előző lekérdezésben meg tudtuk határozni a tulajdonság-számát.

```
//szetszedjuk az utvonalakat
@szetdobott =
    SELECT Item, Paths
    FROM @paths
         CROSS APPLY
         EXPLODE(Paths) AS r(Item);

//leszurom az Item-eket
@osszesit =
    SELECT s.Item, s.Paths
    FROM @szetdobott AS s
         INNER JOIN (SELECT Subject FROM @properties) AS p
         ON s.Item == p.Subject;
```

Minden útvonal egy listaként került tárolásra, melyben az elemek az egyes taxonómiai típusok, az első elem mindenhol az Entity, az elemek sorrendje pedig a

közvetlen subclass of kapcsolatokat jelenti. Eszerint, egy adott típus adott útvonalon található összes őse minden, a listában előtte szereplő elem. Ha az útvonal-listát nullától a szóban forgó típus pozíciójáig indexeljük, éppen a szülőket kapjuk.

```
@indexelve =
    SELECT Item, Paths, Paths.ToList<string>().IndexOf(Item) AS Index
    FROM @osszesit;

@osok =
    SELECT          Item,          Paths,          Index,          new
SqlArray<string>(Paths.ToList<string>().GetRange(0,Index).ToArray())    AS
Parents
    FROM @indexelve;
```

Jelen formában egy típus több sorban is megjelenhet, hiszen több útvonalon is előfordulhat, illetve a hozzájuk tartozó útvonalak összessége sem csak különböző elemeket tartalmaz. Mindkét problémát kiküszöböli az, ha először szétbontjuk az útvonalakat típus – szülő formára, majd csoportosítunk a típusok szerint úgy, hogy a szülő csomópontokból egy egyedi elemeket tartalmazó listát készítünk.

```
@oslistaSzet =
    SELECT Item, Parent
    FROM @osok
         CROSS APPLY
         EXPLODE(Parents) AS r(Parent);

@egyediOs =
    SELECT Item, ARRAY_AGG(DISTINCT Parent) AS Parents
    FROM @oslistaSzet
    GROUP BY Item;
```

Megvan tehát minden típushoz az összes ősosztálya. Ahhoz, hogy a tulajdonság számokat hozzájuk tudjuk illeszteni, megint szét kell szedni a listákat. Az illesztés után egy újabb csoportosítással összegezhetők a szülők tulajdonságai. Végső lépésként pedig hozzá kell adni az összeghez magának a taxonómiai típusnak a tulajdonságait is.

```
@egyediOsSzet =
    SELECT Item, Parent
    FROM @egyediOs
         CROSS APPLY
         EXPLODE(Parents) AS r(Parent);

@osTulajszam =
    SELECT  e.Item  AS  Item,  e.Parent  AS  Parent,  p.Properties.Count  AS
PropNum
    FROM @egyediOsSzet AS e
         INNER JOIN (SELECT Subject, Properties FROM @properties) AS p
         ON e.Parent == p.Subject;

//tipusok szerint csoportositas
@egyediOs =
```

```
    SELECT Item, ARRAY_AGG(DISTINCT Parent) AS Parents, SUM(PropNum) AS
PropSum
    FROM @osTulajszam
    GROUP BY Item;

//ehhez hozza kell meg venni minden Item sajat prop szamat
@join =
    SELECT e.Item, (e.PropSum + p.Properties.Count) AS Properties
    FROM @egyediOs AS e
         INNER JOIN (SELECT Subject, Properties FROM @properties) AS p
         ON e.Item == p.Subject;
```

Táblába írás után a kapott értékeken könnyen meghatározható néhány statisztikai mutató. Eszerint, a taxonómiai típusoknak átlagosan **1100** tulajdonságuk van **743**-mas szórás értékkel. A maximum érték **5408**, a minimum **3**, a medián **975**.

Egy legördülő menüben 1000 különböző tulajdonság közül választani nem egy átlátható feladat. Ezek a hatalmas számok nem meglepők figyelembe véve azt, hogy minden típushoz számításba vettük az összes példányát és azok tulajdonságait. Korábban volt róla szó, hogy a példányok túlnyomó többségben vannak a taxonómiai típusokhoz képest, és ennek oka a tudásbázist szerkesztők viselkedésében keresendő. Hasonlóan, az egyes példányokhoz megkötés nélkül felvehetők tulajdonságok, nincs semmilyen szabály, ami bármit is akadályozna. És ha felvehetők, akkor a felhasználók fel is fogják venni őket, mint azt az elemzés is jól bizonyítja.

3.4.4.2 Séma szerinti vizsgálat

Ha a séma alapján készítjük el az elemzést, akkor is gondolni kell a normál és az inverz tulajdonságokra is. Hasonlóan az előző ponthoz, a végső cél az, hogy egy Taxonómiai típus – Tulajdonságok listája formátumú táblát alakítsunk ki, amin az elemzések már könnyedén elvégezhetők.

Az előre iránynál nem fogunk különösebb nehézségekbe ütközni, hiszen minden taxonómiai típusnak van egy *properties for this type* (P1963) tulajdonsága, ami megmutatja, hogy a típus példányai séma szerint milyen tulajdonságokkal rendelkeznek / rendelkezhetnek. Nincs más dolgunk, mint minden *Taxonómiai típus* – P1963 – PXX alakú állítást megkeresni.

Formálisan:

Tulajdonság_séma_normál := { $\forall p \mid \exists s,o,q: <s, o, q> \in W(P1963), s \in TT, o \in P, q \in (P \times V)$}

U-SQL lekérdezéssel megfogalmazva:

```
@taxTypes =
    SELECT DISTINCT
        Subj
    FROM SubjPropObj
    WHERE Prop == "P279";

@itemProps =
    SELECT Subj, Obj FROM SubjPropProp
    WHERE Prop == "P1963";

@tax_filter =
    SELECT i.Subj AS Subject, i.Obj AS Obj
    FROM @itemProps AS i
         INNER JOIN(
            SELECT Subj FROM @taxTypes) AS t
         ON i.Subj == t.Subj;
```

Az inverz tulajdonságok az egyes tulajdonságokhoz (PYY-hoz) tartozó állításként jelennek meg: ha egy PYY tulajdonság *value type constraint*-je QXX-re mutat, akkor PYY QXX egy inverz tulajdonsága.

Például, az *educated at*[33] (P69) *value type constraint* tulajdonsága *educational institution* (Q2385804), *fictional educational institution* (Q15690029), *university system* (Q20857129), *education program* (Q50433915) típusokra mutat, így ezek mindegyikéhez felveendő P69 mint tulajdonság.

Ezek a konkrét típusok azonban nem egy állítás tárgyaként, hanem úgynevezett *Qualifier*-ként szerepelnek. Adott PYY-nak van egy *property constraint* (P2302) tulajdonsága, ami több mindenre is mutat(hat), köztük a *value type constraint* (Q21510865) típusra. Ennek lehet több, másik tulajdonsága, jelen helyzetben a *class* (P2308) érdekel minket, hiszen ennek tárgyaként szerepelnek majd azon QXX típusok, akikhez a vizsgált tulajdonságot fel kell venni.

Formálisan:

Tulajdonság_séma_inverz := { ∀s,q | ∃<s, Q21510865, q> ϵ W(P2302), s ϵ P, q ϵ (P2308×V)}

U-SQL lekérdezésként megfogalmazva:

```
@inverz_starter = SELECT Id AS Subject, QualifierValues AS Properties
FROM PropertiesWithQualifiers
     WHERE PropConstraint == "P2302" && ValueTypeConstraint == "Q21510865"
&& Class == "P2308";
```

[33] https://www.wikidata.org/wiki/Property:P69

```
@inverz_exploded =
    SELECT de.Subject AS Obj, emp AS Subject
      FROM @inverz_starter AS de
     CROSS APPLY
     EXPLODE(de.Properties) AS dp(emp);

@union =
    SELECT Subject, Obj FROM @tax_filter
    UNION ALL
    SELECT Subject, Obj FROM @inverz_exploded;

@grouping =
    SELECT Subject, ARRAY_AGG(DISTINCT Obj) AS Properties
    FROM @union
    GROUP BY Subject;
```

A végső formátum kialakítása során arra kell figyelni, hogy megegyezzen a példányok szerinti vizsgálatnál használttal, így a statisztikákat előállító lekérdezés módosítás nélkül újra felhasználható.

Ha ebből a szempontból közelítjük meg a kérdést, egy nagyságrenddel kisebb eredményeket kapunk. Átlagosan **107** tulajdonsággal rendelkezik egy típus, **80**-as szórás értékkel. A maximum érték **440**, a minimum **2**, a középérték **94**. Ez már egy sokkal átláthatóbb alkalmazást sugall, kérdés, hogy melyik irány a helyes. A séma szerint megadott tulajdonságok közt nagy valószínűséggel találunk relevánsabbakat, mint azok közt, melyek példányokon szerepelnek. Ha tehát ténylegesen javaslatokat akarunk tenni a felhasználónak, érdemes lehet ezt az irányt követni. Ugyanakkor, a példányok szerinti megközelítés reálisabb képet arról, hogy hány különböző tulajdonsággal ruházhatók fel az egyes entitások.

4 Összefoglalás

A Wikidata napjaink legkiterjedtebb tudásbázisa. Több, mint 50 millió entitásról tartalmaz információt ~700 millió állítás formájában. Szerkesztői nemcsak az egyes területek szakértői, hanem laikus felhasználók és automatizmusok is. Az erős közösségi támogatásnak köszönhetően dinamikusan növekszik, ugyanakkor fokozatosan bonyolódik, hagyományos eszközökkel már át sem látható.

Habár a benne tárolt adatok helyessége és jó minősége fontos szempont, nincs semmilyen validáció az állítások felvitelére vonatkozólag. Emiatt mindig csak az adathalmaz releváns részének ‘nagy része’ fog az elvártnak megfelelően viselkedni, szinte bármilyen minőségi követelményt fogalmaznánk meg, arra találnánk ellenpéldát is.

Jelen tanulmány egyik fő célja az adatbázisról a felhasználók fejében kialakult kép és a valós felépítés összeegyeztetése, közelebb hozása. Ezt az adathalmazban található típusok hasonló jellemzőkön alapuló csoportosításával, a csoportok számosságával, bizonyos tulajdonságok meglétének vizsgálatával, különböző statisztikai mutatók számításával és értelmezésével szeretnénk elérni. Mindazonáltal, az elemzések közvetlen hozadéka, hogy fókuszba kerülnek hibás és hiányos adatok is. A problémák súlyosságának felmérésével javaslatokat tehetünk arra, hogy mely területeken, akár konkrét entitásokon lenne érdemes javítani a tudásbázis minőségének növelése érdekében.

A Wikidata szerkezeti felépítésének középpontjában két kiemelt szereppel rendelkező állítás van: a *subclass of*, amin keresztül az absztrakt típusok kapcsolódnak egymáshoz, és az *instance of*, melynek segítségével kifejezhető, ha egy adott entitás példánya egy típusnak. Először több fogalmat is definiáltunk, mint például *Hivatkozott típus*, *Taxonómiai típus* vagy *Levél*, melyek használatával egyszerűen beszélhetünk bizonyos tulajdonságokkal rendelkező egyedek összességéről. A precíz meghatározás érdekében felhasználtuk a Wikidata egy formális definícióját, segítségével minden fogalmat és lekérdezést formálisan is megadtunk.

Az elemzések során felderítettük, hogy ~58 millió különböző egyed található az adatbázisban. Ezek kisebb része, ~2,4 millió a típusrendszerhez tartozik, míg a többség,

~54,5 millió példányként szerepel. A fennmaradó ~1 millió adatot ebben a megközelítésben nem lehet egyik oldalra sem sorolni, mert nem rendelkeznek hozzá a szükséges állításokkal. Ez nem jelenti azt, hogy teljesen hibásak lennének, csupán kiegészítendők a szükséges állításokkal.

Az ~58 millió egyedből ~60 ezer csak állítások tárgyaként szerepel, azoknak semmilyen tulajdonságuk nincsen. Kialakulásuk tipikus oka lehet az, hogy a szerkesztést végző felhasználó csak hivatkozni szeretett volna rájuk, ám előzőleg nem voltak benn az adatbázisban. Elkészítette hát a hozzájuk tartozó oldalt, létrehozta a hivatkozást, de a részletes kitöltést ráhagyta ‘valaki másra’.

Definíciónk szerint *Hivatkozott típus* lesz minden olyan adattag, amire mutat *instance of* állítás. Ezek száma ~62 ezer, ami meglepő annak fényében, hogy ~54 millió példány van a tudásbázisban. Véleményünk szerint a nagyságrendbeli különbség például a közösségi szerkesztéssel magyarázható, hiszen az emberek többségét konkrét dolgok (emberek, filmek, könyvek) érdeklik, nem absztrakt típusok. Következésképp, az adathalmaz is ezt a preferenciát fogja reflektálni.

További érdekesség az említett ~54 millió példány eloszlása. Egy típushoz átlagosan 1100 tartozik, 109,800-as szórás értékkel. Minimum 1, maximum ~22 millió (!) példánya van egy típusnak, míg a medián 4. Van tehát néhány típus, ami a példányok túlnyomó többségével rendelkezik, míg a maradéknak csupán pár darab van. A lekérdezésből az is látszik, hogy tudományos cikkből (scholarly article, Q13442814) találunk a legtöbbet, az összes entitás közel 40%-át alkotják.

Elméletben az egész tudásbázis gerincét alkotó leszármazási hierarchia egy olyan irányított gráf, ahol a csomópontok a taxonómiai típusok, A és B csúcs között pedig akkor van A -> B irányított él, ha létezik A *subclass of* B állítás. Egy típusnak több közvetlen őse is lehet, és végső soron mindegyik a gyökér típusból, az *Entity*-ből származik. A taxonómiában a *subclass of* állításon keresztül elérhető útvonalak felderítése során szembesültünk azzal, hogy a valóságban ez az elméleti felépítés számtalan körrel egészül ki. Automatizált ellenőrzések körök kialakulása ellen sincsenek, a szerkesztőktől pedig képtelenség lenne elvárni, hogy manuálisan végezzék azt. Ez alapján a körök létezését jelenleg a tudásbázis velejárójának tekintjük, habár a valóságban nehezen értelmezhető, hogy egy típus ősosztálya legyen saját magának.

Végezetül, a Wikidatához hasonlóan nagy méretű adathalmazokkal dolgozó, vizuális lekérdezés-építő alkalmazások nehézségeit vizsgáltuk. Arra a kérdésre kerestük a választ, hogy ha kiválasztunk egy típust a felületen, átlagosan hány különböző tulajdonságot adhatunk meg hozzá? A megoldást két különböző megközelítéssel adtuk meg. Habár alapjaikban különböznek, annyiban megegyeznek, hogy egy típushoz nemcsak a saját tulajdonságai tartoznak, hanem a taxonómiában található szülő csomópontjainak tulajdonságai is.

Először megkerestük a típusokhoz tartozó összes példányt, majd vettük a példányok normál és inverz tulajdonságainkat unióját. A típusrendszerben található útvonalak segítségével előállítottuk az ősosztályokat, ami alapján egy típusnak átlagosan 1100 tulajdonsága van 740-es szórás értékkel, a maximum tulajdonság szám 5408, a minimum 3, a középérték pedig 975. Ezek a hatalmas számok nem kifejezetten meglepőek, hiszen a példányokat a legkülönbözőbb módokon lehet kitölteni, azokhoz gyakorlatilag bármilyen tulajdonságot lehet rendelni. Ugyanakkor 1100 különböző tulajdonságot egy legördülő menüben átlátni lehetetlen feladat.

A másik megközelítésben a séma szerinti tulajdonság számokat vettük alapul. A típusoknak lehet egy ‘Properties for this type’ tulajdonsága, ami több másik tulajdonságra mutat. Az inverz tulajdonságokat is figyelembe véve, eszerint a módszer szerint egy típusnak átlagosan 107 tulajdonsága van 80-as szórás értékkel, a minimum érték 2, a maximum 440, míg a medián 94. Ez már egy sokkal átláthatóbb listát eredményez, és megvan az az előnye, hogy nagy valószínűséggel fontosak és relevánsak is a felkínált tulajdonságok.

Az bemutatott vizsgálatok mellett számos különböző szempontból elemezhetnénk még az adathalmazt. A tulajdonságokat például csak felhasználtuk, azoknak a számát és a hozzájuk tartozó lényeges állításokat nem ismerjük. További állítások alapján még több, egyenként kevesebb elemszámmal rendelkező részekre bonthatnánk az adatbázist, és vizsgálhatnánk az egy csoportba tartozó elemek közös vonásait.

Reméljük, hogy munkánk hasznosnak bizonyul a Wikidata felépítésének megértésében, és ötleteket ad olyan területek, problémák előtérbe helyezésére, melyek javításra szorulnak.

Irodalomjegyzék

[1] L. Galárraga, S. Razniewski, A. Amarilli, and F. M. Suchanek. *Predicting completeness in knowledge bases.* WSDM, 2017.

[2] Vevake Balaraman, Simon Razniewski, Werner Nutt. *Recoin: Relative Completeness in Wikidata.* WWW, 2018.

[3] Michael Luggen, Djellel Difallah, Cristina Sarasua, Gianluca Demartini, Philippe Cudré-Mauroux. *Non-parametric Class Completeness Estimators for Collaborative Knowledge Graphs—The Case of Wikidata.* The Semantic Web – ISWC 2019.

[4] Stefan Heindorf, Martin Potthast, Benno Stein, Gregor Engels. *Vandalism Detection in Wikidata*. CIKM, 2019.

[5] Stefan Heindorf, Martin Potthast, Benno Stein, Gregor Engels. *Towards Vandalism Detection in Knowledge Bases: Corpus Construction and Analysis.* SIGIR, 2015.

[6] Amir Sarabadani, Aaron Halfaker, Dario Taraborelli. *Building Automated Vandalism Detection Tools for Wikidata.* WWW, 2017.

[7] Stefan Heindorf, Yan Scholten, Gregor Engels, Martin Potthast. *Debiasing Vandalism Detection Models at Wikidata*. WWW, 2019.

[8] D. Abián, F. Guerra, J. Martínez-Romanos, Raquel Trillo-Lado. *Wikidata and DBpedia: A Comparative Study.* IKC, 2017.

[9] Färber, Michael; Bartscherer, Frederic; Menne, Carsten; Rettinger, Achim. *Linked data quality of DBpedia, Freebase, OpenCyc, Wikidata, and YAGO.* The Semantic Web, 2018.

[10] F. M. Suchanek, G. Kasneci, and G. Weikum. *YAGO: a core of semantic knowledge.* WWW, 2007.

[11] Mirza, P., Razniewski, S., & Nutt, W. (2016). *Expanding Wikidata's Parenthood Information by 178%, or How To Mine Relation Cardinalities.* In T. Kawamura, & H. Paulheim (Eds.), Proceedings of the ISWC 2016 Posters & Demonstrations Track co-located with 15th International Semantic Web Conference. CEUR-WS.org.

[12] Fredo Erxleben, Michael Günther, Markus Krötzsch, Julian Mendez, Denny Vrandečić. *Introducing Wikidata to the Linked Data Web.* The Semantic Web – ISWC 2014.

[13] Tom Hanika, Maximilian Marx, Gerd Stumme. *Discovering Implicational Knowledge in Wikidata*. ICFCA, 2019.

[14] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, Zachary Ives. DBpedia: *A Nucleus for a Web of Open Data*. ISWC, 2007.

[15] Thomas Pellissier Tanon, Gerhard Weikum, Fabian Suchanek. *YAGO 4: A Reason-able Knowledge Base*. ESWC, 2020.

[16] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, Jamie Taylor. *Freebase: a collaboratively created graph database for structuring human knowledge.* SIGMOD, 2008.

[17] Barabási Albert-László. *Behálózva*. 2002.

Függelék

ItemekFelosztasa.usql

```
@data = SELECT * FROM SubjPropObj;

@allDistSubject =
    SELECT DISTINCT Subj
    FROM @data;

@allDistObj =
    SELECT DISTINCT Obj AS Subj
    FROM @data;

//osszes Item
@osszesitve =
    SELECT Subj FROM @allDistSubject
    UNION DISTINCT
    SELECT Subj FROM @allDistObj;

@osszes_darab =
    SELECT COUNT() AS CCc FROM @osszesitve;

//minden olyan Item, amikrol semmilyen allitas nincsen
@hibas_itemek =
    SELECT *
    FROM @allDistObj
    EXCEPT DISTINCT
    SELECT *
    FROM @allDistSubject;

@hibas_item_darabszam =
    SELECT COUNT() AS cc FROM @hibas_itemek;

OUTPUT @osszes_darab
TO "/wikidata/item-felosztas/osszes-item-darabszam.csv"
USING Outputters.Csv(outputHeader:true,quoting:true);

OUTPUT @hibas_item_darabszam
TO "/wikidata/item-felosztas/hibas-item-darabszam.csv"
USING Outputters.Csv(outputHeader:true,quoting:true);
```

TipusokFelosztasa.usql

```
@data=
    SELECT
        Subj,
        Prop,
        Obj
    FROM SubjPropObj;

//Taxonomiai tipusok
@taxtipus =
    SELECT DISTINCT Subj
    FROM @data
```

```
    WHERE Prop == "P279";

//Hivatkozott tipusok
@hivatkozott =
    SELECT DISTINCT Obj
    FROM @data
    WHERE Prop == "P31";

@taxDeNemHivatkozott =
    SELECT * FROM @taxtipus
    EXCEPT DISTINCT
    SELECT * FROM @hivatkozott;

@countTaxMinuszHiv =
    SELECT COUNT() AS Cc
    FROM @taxDeNemHivatkozott;

@taxEsHiv =
    SELECT * FROM @taxtipus
    INTERSECT DISTINCT
    SELECT * FROM @hivatkozott;

@countTaxEsHiv =
    SELECT COUNT() AS Cc
    FROM @taxEsHiv;

@allDistSubject =
    SELECT DISTINCT Subj
    FROM @data;

@allDistObj =
    SELECT DISTINCT Obj AS Subj
    FROM @data;

//osszes Item
@filter =
    SELECT Subj FROM @allDistSubject
    UNION DISTINCT
    SELECT Subj FROM @allDistObj;

@union =
    SELECT * FROM @taxtipus
    UNION
    SELECT * FROM @hivatkozott;

@egyiksem =
    SELECT * FROM @filter
    EXCEPT DISTINCT
    SELECT * FROM @union;

@countEgyiksem =
    SELECT COUNT() AS Cc
    FROM @egyiksem;

OUTPUT @countTaxMinuszHiv
TO "/wikidata/taxonomiai-de-nem-hivatkozott-darabszam.csv"
USING Outputters.Csv(outputHeader:true,quoting:true);

OUTPUT @countTaxEsHiv
```

```
TO "/wikidata/taxonomiai-es-hivatkozott-darabszam.csv"
USING Outputters.Csv(outputHeader:true,quoting:true);

OUTPUT @countEgyiksem
TO "/wikidata/nem-taxonomiai-es-nem-hivatkozott-darabszam.csv"
USING Outputters.Csv(outputHeader:true,quoting:true);
```

HivatkozottTipusPeldany.usql

```
@subjPropObj=
    SELECT
        Subj,
        Prop,
        Obj
    FROM SubjPropObj;

//szerepel instanceOf allitas alanyakent
@alanyok =
    SELECT DISTINCT Subj FROM @subjPropObj
    WHERE Prop == "P31";

//van subclassOf allitasa
@van =
    SELECT DISTINCT Subj FROM @subjPropObj
    WHERE Prop == "P279";

@peldanyok =
    SELECT * FROM @alanyok
    EXCEPT
    SELECT * FROM @van;

@count =
SELECT COUNT() AS ccc FROM @peldanyok;

@sample=
    SELECT *
    FROM @peldanyok
         SAMPLE ANY(30);


OUTPUT @count
TO "/wikidata/hivatkozott_tipus_peldany.csv"
USING Outputters.Csv(outputHeader:true,quoting:true);

OUTPUT @sample
TO "/wikidata/hivatkozott_tipus_peldany-minta.csv"
USING Outputters.Csv(outputHeader:true,quoting:true);
```

HianyVizsgalat.usql

```
@data =
    SELECT *
    FROM SubjPropObj;

@allDistSubject =
    SELECT DISTINCT Subj
    FROM @data;

@allDistObj =
```

```
    SELECT DISTINCT Obj AS Subj
    FROM @data;

//osszes Item
@osszesitem =
    SELECT Subj FROM @allDistSubject
    UNION DISTINCT
    SELECT Subj FROM @allDistObj;

//kiknek nincs instance Of allitasa?
//kiknek van
@van_instanceof = SELECT DISTINCT Subj FROM @data WHERE Prop == "P31";
//instanceOf

//nincs = osszes-van
@nincs_instanceof =
    SELECT Subj FROM @osszesitem
    EXCEPT DISTINCT
    SELECT * FROM @van_instanceof;

@instance_hiany_darab =
    SELECT COUNT() AS Ccc
    FROM @nincs_instanceof;

//nincs SubclassOf allitasa
//kiknek van
@van_subclass =
    SELECT DISTINCT Subj
    FROM @data
    WHERE Prop == "P279";

//nincs = osszes - van
@nincs_subclass =
    SELECT Subj FROM @osszesitem
    EXCEPT
    SELECT Subj FROM @van_subclass;

@subclass_hiany_darab =
    SELECT COUNT() AS Ccc FROM @nincs_subclass;

//se instance se subclass
@egyiksem =
    SELECT * FROM @nincs_instanceof
    INTERSECT
    SELECT * FROM @nincs_subclass;

@egyiksem_darab =
    SELECT COUNT() AS Ccc
    FROM @egyiksem;

@egyiksem_minta =
SELECT * FROM @egyiksem SAMPLE ANY(20);

//nincs subclassOf-ja, de van olyan subclassOf, ami ra mutat
@filter =
    SELECT DISTINCT Obj FROM @data AS d
                            INNER JOIN (SELECT Subj FROM @nincs_subclass)
AS n
                            ON d.Obj == n.Subj
```

```
    WHERE d.Prop == "P279";

@filter_darab =
    SELECT COUNT() AS Ccc FROM @filter;

@filter_minta =
    SELECT * FROM @filter SAMPLE ANY(20);



OUTPUT @instance_hiany_darab
TO "/wikidata/hiany-vizsgalat/item-instanceOf-nelkul-darabszam.csv"
USING Outputters.Csv(outputHeader:true,quoting:true);

OUTPUT @subclass_hiany_darab
TO "/wikidata/hiany-vizsgalat/item-subclassOf-nelkul-darabszam.csv"
USING Outputters.Csv(outputHeader:true,quoting:true);

OUTPUT @egyiksem_darab
TO            "/wikidata/hiany-vizsgalat/item-subclassOf-instanceOf-nelkul-
darabszam.csv"
USING Outputters.Csv(outputHeader:true,quoting:true);

OUTPUT @egyiksem_minta
TO "/wikidata/hiany-vizsgalat/item-subclassOf-instanceOf-nelkul-minta.csv"
USING Outputters.Csv(outputHeader:true,quoting:true);

OUTPUT @filter_darab
TO          "/wikidata/hiany-vizsgalat/nincs-subclass-de-mutat-ra-subclass-
darabszam.csv"
USING Outputters.Csv(outputHeader:true,quoting:true);

OUTPUT @filter_minta
TO          "/wikidata/hiany-vizsgalat/nincs-subclass-de-mutat-ra-subclass-
minta.csv"
USING Outputters.Csv(outputHeader:true,quoting:true);
```

ExtractItemsWithLabels.usql

```
@RawData =
    SELECT [Id]
    FROM WikidataImport
    WHERE Type == "item";

@tablaban =
    SELECT DISTINCT Subj FROM SubjPropObj
    UNION
    SELECT DISTINCT Obj FROM SubjPropObj;

@except =
    SELECT * FROM @RawData
    EXCEPT
    SELECT * FROM @tablaban;



@sample =
SELECT * FROM @except SAMPLE ANY(20);
```

```
@darab =
SELECT COUNT() AS Ccc FROM @except;
/*
INSERT INTO PropPropSubj
SELECT
    Subj,
    Prop,
    Obj
FROM @jsonPathResultExploded;
*/

OUTPUT @sample
TO "/wikidata/arva-item-minta.csv"
USING Outputters.Csv(outputHeader:true,quoting:true);

OUTPUT @darab
TO "/wikidata/arva-item-darabszam.csv"
USING Outputters.Csv(outputHeader:true,quoting:true);
```

CalculateDepth.usql

```
//ket reszbol all
//elso resz az adatok eloallitasa, masodik az elemzesuk

//elso resz
/*
@data = SELECT *
                FROM (VALUES
                          ("Q2", "P1", "Q1")
                        , ("Q3", "P1", "Q1")
                        , ("Q4", "P1", "Q1")
                        , ("Q5", "P1", "Q2")
                        , ("Q6", "P1", "Q2")
                        , ("Q7", "P1", "Q2")
                        , ("Q8", "P1", "Q3")
                        , ("Q9", "P1", "Q6")
                        , ("Q10", "P1", "Q6")
                        , ("Q11", "P1", "Q8")
                        , ("Q12", "P1", "Q9")
                        , ("Q7", "P1", "Q3")
                        , ("Q10", "P1", "Q8")
                        , ("Q12", "P1", "Q5")
                        , ("Q1", "P1", "Q11")
                        , ("Q3", "P1", "Q11")
                        , ("Q5", "P1", "Q4") ) AS D(Subj, Prop, Obj);
*/

@data =
    SELECT
        Subj,
        Prop,
        Obj
    FROM SubjPropObj
    WHERE Prop == "P279";

DROP FUNCTION IF EXISTS dbo.SimpleRecursion;
CREATE   FUNCTION   dbo.SimpleRecursion(@path   TABLE(LastElement   string,
CircleFlag   bool,   Paths   SqlArray<string>),   @founded   TABLE(LastElement
```

```
string, CircleFlag bool, Paths SqlArray<string>), @pathLen int = 0,
@MaxIterations int = 5, @data TABLE(Subj string, Prop string, Obj string))
RETURNS @result TABLE(LastElement string, CircleFlag bool, Paths
SqlArray<string>)
AS
BEGIN

    @left_join =
        SELECT f.Subj AS LastElement, p.Paths.Contains(f.Subj) AS
CircleFlag, new SqlArray<string>(p.Paths){f.Subj} AS Paths
               FROM (SELECT * FROM @path WHERE CircleFlag == true AND
LastElement != null) AS p
                    LEFT OUTER JOIN (SELECT Subj, Obj FROM @data) AS f
                                    ON f.Obj == p.LastElement;

    @results =
        SELECT * FROM @founded
        UNION ALL
        SELECT * FROM @path WHERE LastElement == null;

    IF (@pathLen >= @MaxIterations) THEN
        @result = SELECT * FROM @results
        UNION ALL
        SELECT * FROM @left_join;
    ELSE
        @result = dbo.SimpleRecursion(@left_join, @results, @pathLen + 1,
@MaxIterations, @data);
    END;


END;

//@path = SELECT * FROM (VALUES ("Q1", false, new SqlArray<string>(new
string [] {"Q1"})) ) AS T(LastElement, CircleFlag, Paths);
//@founded = SELECT * FROM (VALUES ("Q1", false, new SqlArray<string>(new
string [] {"Q1"})) ) AS T(LastElement, CircleFlag, Paths);

@path = SELECT * FROM (VALUES ("Q35120", true, new SqlArray<string>(new
string [] {"Q35120"})) ) AS T(LastElement, CircleFlag, Paths);
@founded = SELECT * FROM (VALUES ("init", true, new SqlArray<string>(new
string [] {"init"})) ) AS T(LastElement, CircleFlag, Paths);

//@path = SELECT * FROM WikidataPaths1;

@toexplode = dbo.SimpleRecursion(@path, @founded, 0, 30, @data);

INSERT INTO WikidataPaths
SELECT
    LastElement,
    CircleFlag,
    Paths
FROM
    @toexplode;



//masodik resz - elemzes
/*
@data =
```

```
    SELECT * FROM WikidataPaths;


@analize =
    SELECT MIN(d.Paths.Count) AS MinValue, MAX(d.Paths.Count) AS MaxValue,
AVG(d.Paths.Count) AS Average, STDEV(d.Paths.Count) AS StdDev
    FROM @data AS d
    WHERE CircleFlag == false;


OUTPUT @analize
TO "/wikidata/paths_length_MIN_MAX_AVG_STDEV.csv"
USING Outputters.Csv(outputHeader:true,quoting:true);*/
```

LevelekSzama.usql

```
//level: van subclass of allitasa, de nem szerepel subclass of targyakent
@data =
    SELECT *
    FROM SubjPropObj;

//meghatarozas: taxonomiai es (osszes - szerepel subclass of targyakent)
@taxonomiai =
    SELECT DISTINCT Subj FROM @data
    WHERE Prop == "P279";

@szerepel =
    SELECT DISTINCT Obj FROM @data
    WHERE Prop == "P279";

@alls =
    SELECT DISTINCT Subj FROM @data;

@allo =
    SELECT DISTINCT Obj FROM @data;

@itemek =
    SELECT * FROM @alls
    UNION
    SELECT * FROM @allo;

@nem_szerepel =
    SELECT * FROM @itemek
    EXCEPT
    SELECT * FROM @szerepel;

@levelek =
    SELECT *
    FROM @taxonomiai
    INTERSECT
    SELECT *
    FROM @nem_szerepel;

@minta =
    SELECT * FROM @levelek
                SAMPLE ANY(20);

@darab =
    SELECT COUNT() AS Ccc FROM @levelek;
```

```
OUTPUT @minta
TO "/wikidata/levelek-minta2.csv"
USING Outputters.Csv(outputHeader:true,quoting:true);

OUTPUT @darab
TO "/wikidata/levelek-darabszam.csv"
USING Outputters.Csv(outputHeader:true,quoting:true);
```

AveragePropOfType.usql

```
//elso resz a taxonomiai tipusokhoz tartozo property-k meghatarozasa
//masodik resz a kapott adathalmaz elemzese

//elso resz - adatok elokeszitese
/*
@subjPropObj =
SELECT * FROM SubjPropObj;

//taxonomiai tipusok
@taxTypes =
    SELECT DISTINCT
        Subj
    FROM @subjPropObj
    WHERE Prop == "P279";

//instance of sorok
@instanceOfRows =
    SELECT * FROM @subjPropObj
    WHERE Prop == "P31";

//tax tipusok peldanyai
//peldany: van instance of de nincs subclass of allitasa
//elso lepes: valami-instance of-taxonomiai tipus
@filter =
    SELECT s.Subj AS Instance, s.Obj AS Types
    FROM @instanceOfRows AS s
         INNER JOIN
         (
         SELECT Subj
         FROM @taxTypes
               ) AS t
         ON s.Obj == t.Subj;

//masodik lepes: nincs subclass-instance of-taxonomiai tipus
//ehhez kell egy tabla, amiben azokat tarolon, akiknek nincsen subclass
of-ja
//legegyszerubb: osszes - van
@nincs_subclass =
    SELECT DISTINCT Subj FROM @subjPropObj
    EXCEPT
    SELECT * FROM @taxTypes;

@instancesOfTaxTypes =
    SELECT f.Instance, f.Types FROM @filter AS f
                                    INNER    JOIN    (SELECT    Subj    FROM
@nincs_subclass) AS n
                                    ON f.Instance == n.Subj;
```

```
//ezen peldanyok propertijei
@propsOfInstances =
    SELECT s.Subj, s.Prop
    FROM @subjPropObj AS s
         INNER JOIN (SELECT Instance FROM @instancesOfTaxTypes) AS i
         ON s.Subj == i.Instance;

//azon prop, ahol tax tipus peldanya az obj
//ami itt Obj, nekem Subj-kent kell az uniohoz
//az elozovel azonos formara hozom
@inverzPropOfInstances =
    SELECT s.Obj AS Subj, s.Prop
    FROM @subjPropObj AS s
         INNER JOIN (SELECT Instance FROM @instancesOfTaxTypes) AS i
                    ON s.Obj == i.Instance;

@unionProperties =
    SELECT * FROM @propsOfInstances
    UNION
    SELECT * FROM @inverzPropOfInstances;

//taxonomiai tipusonkent csinalok egy group-by-t
//a hozzajuk tartozo property-k egy sqlArray-ben
@output =
    SELECT s.Types AS Obj, ARRAY_AGG(DISTINCT p.Prop) AS Prop
    FROM @unionProperties AS p
         INNER JOIN (SELECT Instance, Types FROM @instancesOfTaxTypes) AS
s
         ON s.Instance == p.Subj
    GROUP BY s.Types;

INSERT INTO TaxTypesAndProperties2
SELECT
    Obj,
    Prop
FROM
    @output;
*/

//masodik resz - elemzes

@data =
    SELECT Obj, Prop, 1 AS Same FROM TaxTypesAndProperties2;


@analize =
    SELECT MIN(d.Prop.Count) AS MinValue, MAX(d.Prop.Count) AS MaxValue,
AVG(d.Prop.Count) AS Average, STDEV(d.Prop.Count) AS StdDev
    FROM @data AS d;

@median =
    SELECT DISTINCT Same, PERCENTILE_DISC(0.5) WITHIN GROUP(ORDER BY
d.Prop.Count) OVER(PARTITION BY Same) AS Median
    FROM @data AS d;



OUTPUT @analize
TO "/wikidata/taxonomiai_tipusok_property_MIN_MAX_AVG_STDEV.csv"
```

```
USING Outputters.Csv(outputHeader:true,quoting:true);


OUTPUT @median
TO "/wikidata/taxonomiai_tipusok_property_MEDIAN.csv"
USING Outputters.Csv(outputHeader:true,quoting:true);
```

ItemOsosztalyokEloallitasa.usql

```
//az utvonalak, amikbol ki akarom nyerni
@paths =
    SELECT * FROM WikidataPaths;

//a tabla, amiben ki vannak gyujtve az Itemek, mellette a Property tomb
oszloppal
//ez a sema szerintiekre vonatkozik
/*
@properties =
    SELECT *
    FROM TaxTypesAndPropertiesByScheme;
*/

//ez pedig a peldanyadatok szerintire
@properties =
    SELECT Obj AS Subject, Prop AS Properties
    FROM TaxTypesAndProperties2;

//szetdobom a tombot, ugy, hogy a tomb is megjelenik minden sorban
//ezaltal tudom, hogy melyik elem melyik utvonalhoz tartozik es az utvonal
sorrend sem veszik el
@szetdobott =
    SELECT Item, Paths
    FROM @paths
         CROSS APPLY
         EXPLODE(Paths) AS r(Item);

//leszurom az Item-eket ugy, hogy csak azok maradjanak meg, amik engem
erdekelnek,
//vagyis tudom, hogy hany property-juk van
@osszesit =
    SELECT s.Item, s.Paths
    FROM @szetdobott AS s
         INNER JOIN (SELECT Subject FROM @properties) AS p
         ON s.Item == p.Subject;

//megkeresem, hogy az adott Item hanyadik helyen all
//minden elotte levo elem az o ose lesz, ki tudom igy nyerni
@indexelve =
    SELECT Item, Paths, Paths.ToList<string>().IndexOf(Item) AS Index
    FROM @osszesit;

//megindexelem vele a tombot, kinyerem az osoket
@osok =
    SELECT           Item,           Paths,           Index,           new
SqlArray<string>(Paths.ToList<string>().GetRange(0,Index).ToArray())     AS
Parents
    FROM @indexelve;
```

```
//ekkor van egy Item (ami engem erdekel), Os_tomb tablam
//egy adott Item tobbszor is szerepelhet, hiszen tobb utvonalon is elofordulhat
//azt akarom elerni, hogy minden Item egyszer szerepeljen, es egy DISTINCT os tomb legyen mellette
//Ehhez eloszor szetdobom az osoket
@oslistaSzet =
    SELECT Item, Parent
    FROM @osok
         CROSS APPLY
         EXPLODE(Parents) AS r(Parent);

//csinalok egy group by-t, ahol kikotom, hogy egyedi elemek legyenek az osok kozott
@egyediOs =
    SELECT Item, ARRAY_AGG(DISTINCT Parent) AS Parents
    FROM @oslistaSzet
    GROUP BY Item;

//utana ezt megint szetdobom, mert hozza kell meg rendelni az osokhoz az o sajat property szamukat
@egyediOsSzet =
    SELECT Item, Parent
    FROM @egyediOs
         CROSS APPLY
         EXPLODE(Parents) AS r(Parent);

//join-olom a properties tablat, ezaltal azok az osok maradnak meg, akikrol tenyleg tudok is valamit, es azt a valamit fel is veszem hozzajuk
@osTulajszam =
    SELECT e.Item AS Item, e.Parent AS Parent, p.Properties.Count AS PropNum
    FROM @egyediOsSzet AS e
         INNER JOIN (SELECT Subject, Properties FROM @properties) AS p
         ON e.Parent == p.Subject;

//az igy letrejott adatokat Item szerint csoportositom, osokbol csinalok egy listat, a hozzajuk tartozo prop szamot pedig osszegzem
@egyediOs =
    SELECT Item, ARRAY_AGG(DISTINCT Parent) AS Parents, SUM(PropNum) AS PropSum
    FROM @osTulajszam
    GROUP BY Item;

//ez nem eleg meg
//ehhez hozza kell meg venni minden Item sajat prop szamat
@join =
    SELECT e.Item, (e.PropSum + p.Properties.Count) AS Properties
    FROM @egyediOs AS e
         INNER JOIN (SELECT Subject, Properties FROM @properties) AS p
         ON e.Item == p.Subject;

//es itt vagyok keszen
INSERT INTO PropsByInstancesWithParents
SELECT
    *
FROM
    @join;
```

AvgInstanceOfRealTypes.usql

```
@subjPropObj=
    SELECT
        Subj,
        Prop,
        Obj
    FROM SubjPropObj;

//taxonomiai tipusok - van subclass of-ja
@allTypes=
    SELECT DISTINCT Subj
    FROM @subjPropObj
    WHERE Prop == "P279";

//hivatkozott tipusok - mutat ra instance of allitas
@typesWithInstance=
    SELECT DISTINCT Obj
    FROM @subjPropObj
    WHERE Prop == "P31";

//valodi tipus: hivatkozott es taxonomiai is
@realTypes=
    SELECT Subj AS Obj FROM @allTypes
    INTERSECT DISTINCT
    SELECT * FROM @typesWithInstance;

//peldany: van instance of , de nincs subclass of allitasa
//megkeressuk azokat a sorokat, amik peldany-p31-valodi_tipus formaban
vannak
//peldanyokhoz kell: nincs subclass of-ja
//ezt legegyszerubben: osszes - van
@van =
    SELECT DISTINCT Subj FROM @subjPropObj
    WHERE Prop == "P279";

@nincs_subclass =
    SELECT DISTINCT Subj FROM @subjPropObj
    EXCEPT
    SELECT * FROM @van;

//subjPropObj-et eloszor leszurjuk a valodi tipusokra
//itt mar szurunk arra is, hogy csak instance of allitasaink legyenek
@valodira_mutato_allitasok =
    SELECT s.Subj, s.Prop, s.Obj FROM @subjPropObj AS s
                                INNER JOIN (SELECT Obj FROM @realTypes) AS
r
                                ON s.Obj == r.Obj
    WHERE s.Prop == "P31";

//aztan meghagyjuk azokat, akiknek nincsen subclass of-ja
@filtered =
    SELECT v.Subj, v.Prop, v.Obj FROM @valodira_mutato_allitasok AS v
                INNER JOIN (SELECT Subj FROM @nincs_subclass) AS n
                ON v.Subj == n.Subj;


//csinalunk ra egy group by-t
//Same oszlop a medianhoz kell
```

```
@countOccurances =
    SELECT Obj, COUNT() AS Occurance, 1 AS Same
    FROM @filtered
         GROUP BY Obj;

@jellemzok =
    SELECT   MAX(Occurance)   AS   Maximum,   MIN(Occurance)   AS   Minimum,
AVG(Occurance) AS Average, STDEV(Occurance) AS Spread
    FROM @countOccurances;

@median =
    SELECT  DISTINCT  Same,  PERCENTILE_DISC(0.5)  WITHIN  GROUP(ORDER  BY
Occurance) OVER(PARTITION BY Same) AS Median
    FROM @countOccurances;


//Little filter to find which subject has the most instance
/*
@maxobject =
    SELECT Subj
    FROM @countOccurances
    WHERE Occurance == 22641732;
*/


OUTPUT @jellemzok
TO "/wikidata/valodi-tipusok-peldanyai-AVG-MIN-MAX-STDEV.csv"
USING Outputters.Csv(outputHeader:true,quoting:true);

OUTPUT @median
TO "/wikidata/valodi-tipusok-peldanyai-MEDIAN.csv"
USING Outputters.Csv(outputHeader:true,quoting:true);
```