Practical SPARQL

EGOR DMITRIEV

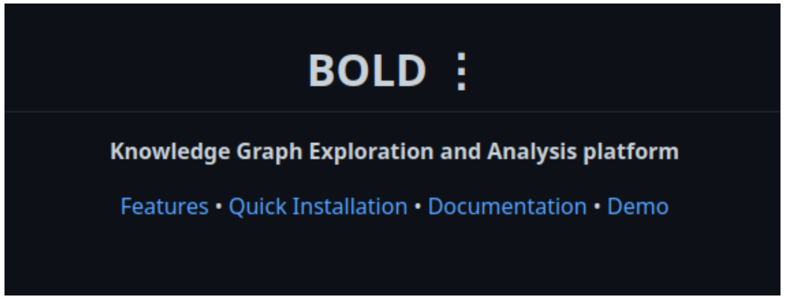
Agenda

- Introduction to BOLD
  - About the project and challenges
- Triple Stores
  - How triples are stored and how to choose a database
- SPARQL
  - Short recap and introduction of services
- Querying Wikidata
  - Intrinsics of wikidata dataset design and how to use it
- Query Performance
  - Tips and tricks to speed up your queries

Resources and Queries: https://github.com/EgorDm/INFOMKDE/

Introduction to BOLD

- Goal: Speed up exploration and bias analysis for knowledge graphs



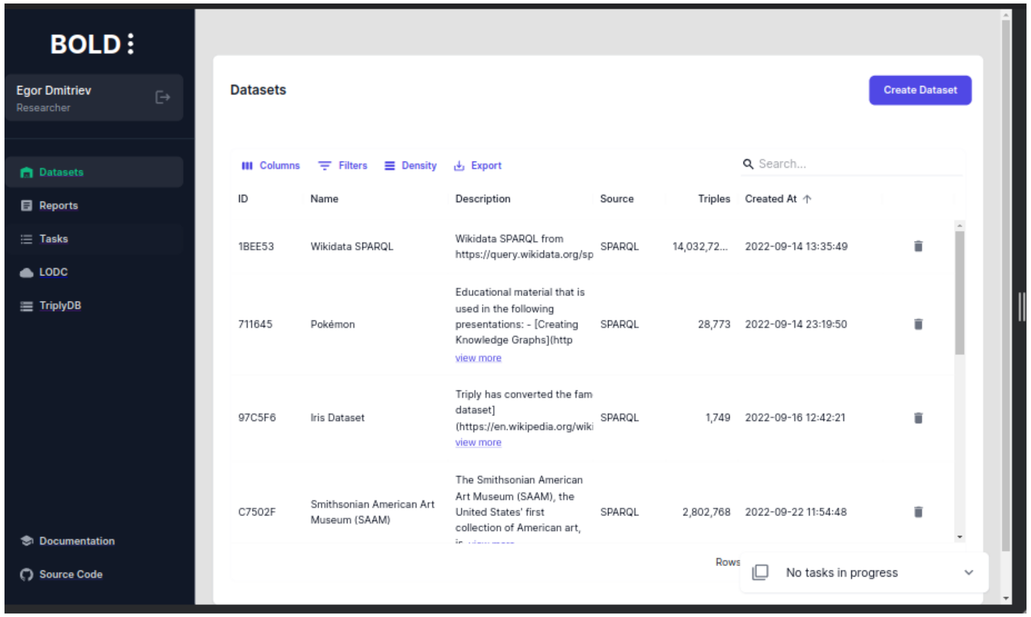
BOLD ⋮

Knowledge Graph Exploration and Analysis platform

Features • Quick Installation • Documentation • Demo

Demo

- https://bold-demo.ml/
- User: demo
- Pass: demodemo
- For wikidata use:
- https://egordm.github.io/BOLDER/

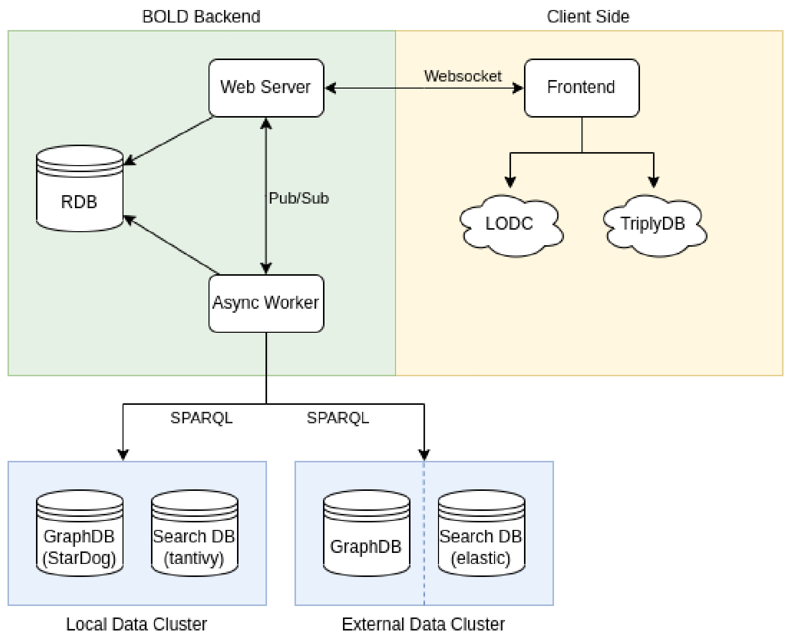


Datasets

ID	Name	Description	Source	Triples	Created At
1BEE53	Wikidata SPARQL	Wikidata SPARQL from https://query.wikidata.org/sp	SPARQL	14,032,72...	2022-09-14 13:35:49
711645	Pokémon	Educational material that is used in the following presentations: - [Creating Knowledge Graphs](http view more	SPARQL	28,773	2022-09-14 23:19:50
97C5F6	Iris Dataset	Triply has converted the fam dataset] (https://en.wikipedia.org/wiki view more	SPARQL	1,749	2022-09-16 12:42:21
C7502F	Smithsonian American Art Museum (SAAM)	The Smithsonian American Art Museum (SAAM), the United States' first collection of American art,	SPARQL	2,802,768	2022-09-22 11:54:48

Challenge 1: Long running queries

- Querying and importing is treated as an asynchronous job
- Data Tier is separate from the orchestartion



Challenge 2: Data Sources

LINKED OPEN DATA CLOUD (LODC)

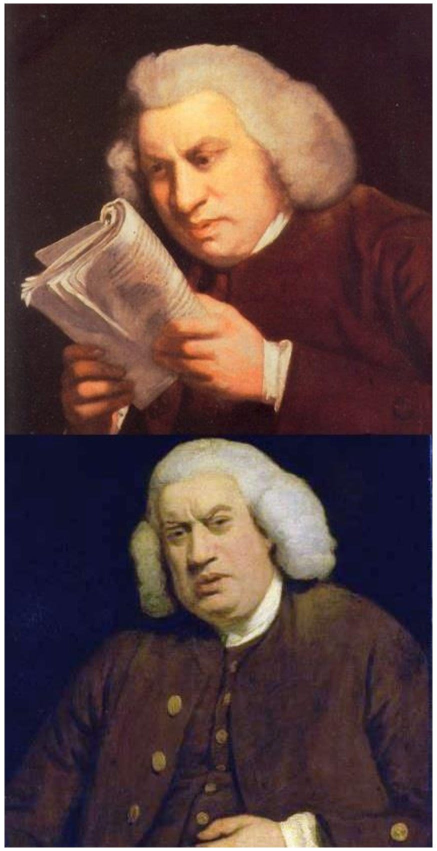
- Free registry for knowledge graphs
- More than 5k listings
- Only 5% works

TRIPLY DB

- Like GitHub for Knowledge Graphs
- Free storage for reasonably sized graphs
- Free virtuoso instance
- Has an API for querying and downloading graphs

Challenge 3: Indexing

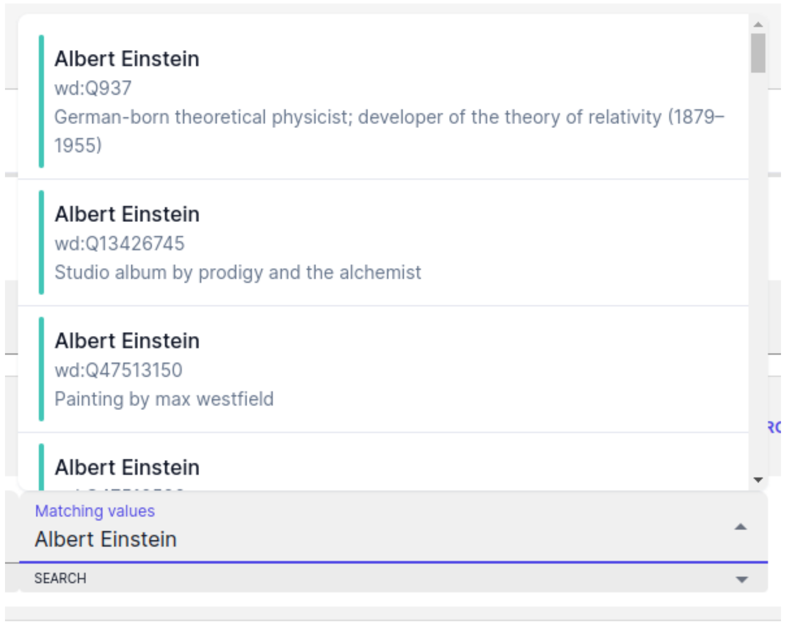
```
SELECT * { # Wikidata
    wd:Q937 wdt:569 ?o
}

SELECT * { # OpenSanctions
    entity:ofac-bc8451a
    w3ftm:Sanction:entity
    ?o
}
```



[1] Johnson (1775) showing his intense concentration as he tries to decipher Wikidata queries

Challenge 3: Indexing

- How do you know what terms you can use?
- Create a full text search index using
  - Embedded search index (tantivy)
  - Search db (elasticsearch)
  - Built-in functionality[1]



Albert Einstein
wd:Q937
German-born theoretical physicist; developer of the theory of relativity (1879–1955)

Albert Einstein
wd:Q13426745
Studio album by prodigy and the alchemist

Albert Einstein
wd:Q47513150
Painting by max westfield

Albert Einstein

[1] https://github.com/blazegraph/database/wiki/fulltextsearch

Triple Stores

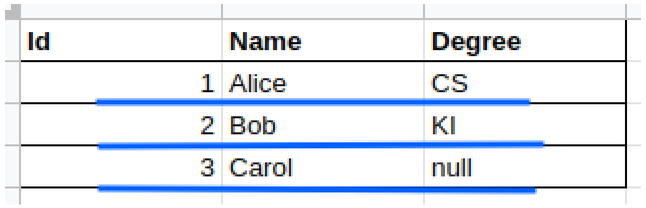
HOW TRIPLES ARE STORED AND HOW TO CHOOSE A DATABASE

What is a Triple Store?

- "A triple store is a database for storage and retrieval of triples through semantic queries"
- Considered as a class of NoSQL databases
  - Data is schemaless
  - Data is not stored in relational tables

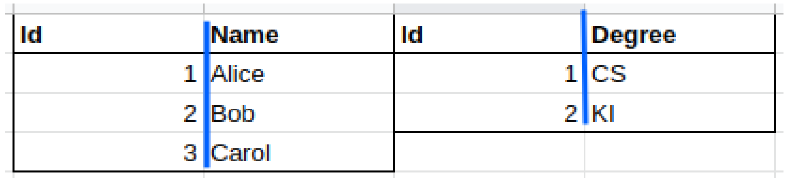
How are triples stored

- Different storage reflects different needs
- Rows storage excels at retrieving complete documents
- Col storage is more efficient for analytical queries and fast storage
- Triples allow to model very complex networks



Id	Name	Degree
1	Alice	CS
2	Bob	KI
3	Carol	null

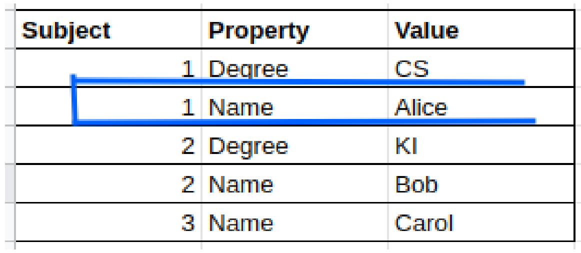
Row storage



Id	Name
1	Alice
2	Bob
3	Carol

Id	Degree
1	CS
2	KI

Column storage

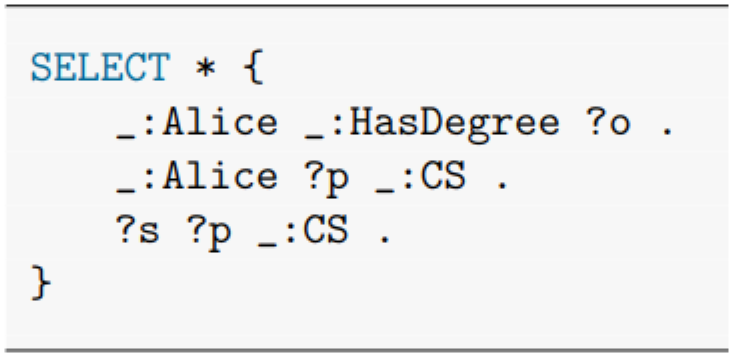


Subject	Property	Value
1	Degree	CS
1	Name	Alice
2	Degree	KI
2	Name	Bob
3	Name	Carol

Triple storage

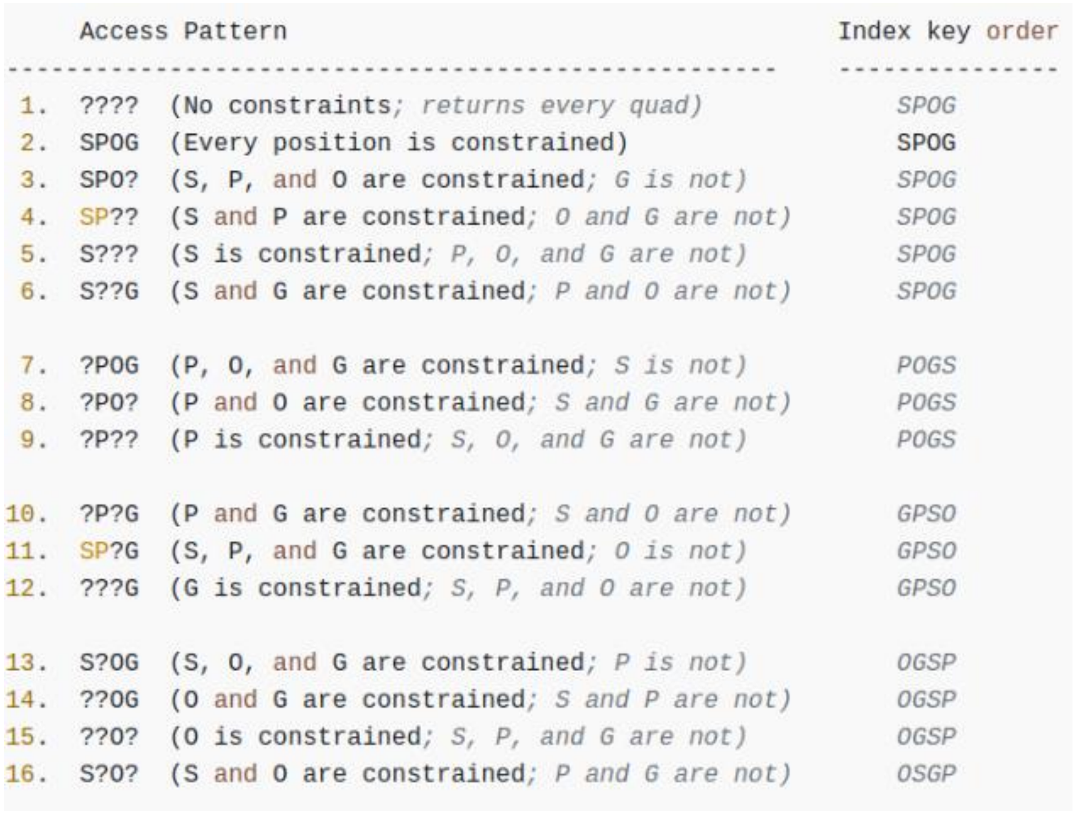
Why are Triple Stores efficient?

- Triple queries have patterns with unconstrained term positions (variables)
- Each pattern involves a join
- Indexing for different access patterns speeds up the search for suitable values



```
SELECT * {
    _:Alice _:HasDegree ?o .
    _:Alice ?p _:CS .
    ?s ?p _:CS .
}
```

Access pattern indexes



```
    Access Pattern                                        Index key order
----------------------------------------------------    ---------------
 1.  ????  (No constraints; returns every quad)              SPOG
 2.  SPOG  (Every position is constrained)                   SPOG
 3.  SPO?  (S, P, and O are constrained; G is not)           SPOG
 4.  SP??  (S and P are constrained; O and G are not)        SPOG
 5.  S???  (S is constrained; P, O, and G are not)           SPOG
 6.  S??G  (S and G are constrained; P and O are not)        SPOG

 7.  ?POG  (P, O, and G are constrained; S is not)           POGS
 8.  ?PO?  (P and O are constrained; S and G are not)        POGS
 9.  ?P??  (P is constrained; S, O, and G are not)           POGS

10.  ?P?G  (P and G are constrained; S and O are not)        GPSO
11.  SP?G  (S, P, and G are constrained; O is not)           GPSO
12.  ???G  (G is constrained; S, P, and O are not)           GPSO

13.  S?OG  (S, O, and G are constrained; P is not)           OGSP
14.  ??OG  (O and G are constrained; S and P are not)        OGSP
15.  ??O?  (O is constrained; S, P, and G are not)           OGSP
16.  S?O?  (S and O are constrained; P and G are not)        OSGP
```

- Entities
  - (S) Subject
  - (P) Predicate
  - (O) Object
  - (G) Graph
- Each triple/quad is characterized by SPOG key
  - (Alice, hasDegree, CS, wikidata)
- Which index to use?
  - (Alice, hasDegree, ?, wikidata)

[1] https://docs.aws.amazon.com/neptune/latest/userguide/feature-overview-data-model.html

What to look for in a Triple Store?

- Does it support your use case?
  - What do you need?
  - Is it maintained or fit your budget?

1. Query languages it supports
   - SPARQL for triple queries
   - Gremlin / Cypher for path or graph queries
2. Index coverage / storage tradeoff
3. Full-text indexing / search
4. Tool integration
   - API / RDF import formats
   - User interface

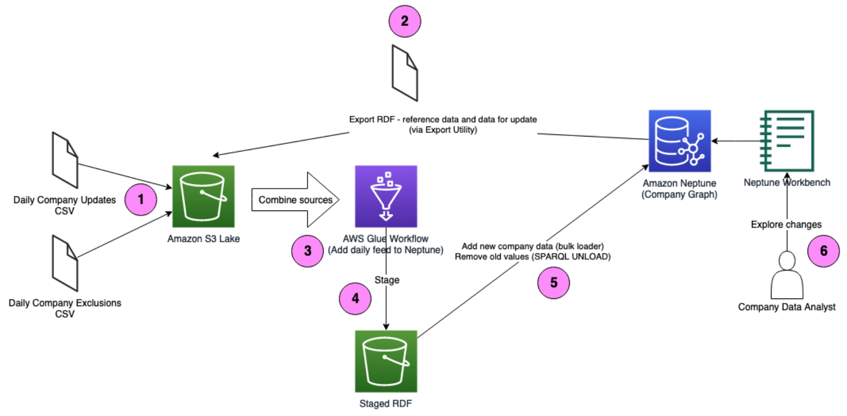
Blazegraph

- Open Source (but no longer maintained)
- Supports SPARQL and Gremlin
- POS, SPO, OSP indexes [1]
- Supports sharding, full text search indexes
- Great Java API for writing extensions

[1] https://github.com/blazegraph/database/blob/master/bigdata-core/bigdata-rdf/src/java/com/bigdata/rdf/spo/SPOKeyOrder.java

AWS Neptune

- Proprietary (fork of Blazegraph)
- Cloud Only
- Supports SPARQL, Gremlin and openCypher
- SPOG, POGS, GPSO, (optional OSGP in lab mode) [1]
- Good integration with AWS ecosystem (S3, sagemaker, etc)


2
Export RDF - reference data and data for update (via Export Utility)
Daily Company Updates CSV
1
Amazon S3 Lake
Combine sources
AWS Glue Workflow (Add daily feed to Neptune)
3
Daily Company Exclusions CSV
Stage
4
Staged RDF
Add new company data (bulk loader) Remove old values (SPARQL UNLOAD)
5
Amazon Neptune (Company Graph)
Neptune Workbench
Explore changes
6
Company Data Analyst

[1] https://docs.aws.amazon.com/neptune/latest/userguide/feature-overview-data-model.html

Stardog

- Proprietary (has a free plan)
- Supports SPARQL, GraphQL and its own path query language
- Maintains 15 out of 16 indexes [1]
- Optimized for high write (and import) throughput
- Supports full text search (only on labelled data)
- Best user interface / tooling

[1] https://www.stardog.com/blog/stardog-in-the-storage-wars/

Virtuoso

- Proprietary (has open source version)
- Supports SPARQL and SQL
- PSOG, POSG, SP, OP, GS indexes [1]
- Overall fastest query performance

[1] Andreas Harth et al. "Linked Data Management: Principles and Techniques"

Others

- GraphDB, AnzoGraph, AllegroGraph, MarkLogic, Apache Rya, Oxigraph
- Have too small community, or are abandoned
- Neo4J supports triples but is a Graph database

SPARQL

SHORT RECAP AND INTRODUCTION OF SERVICES

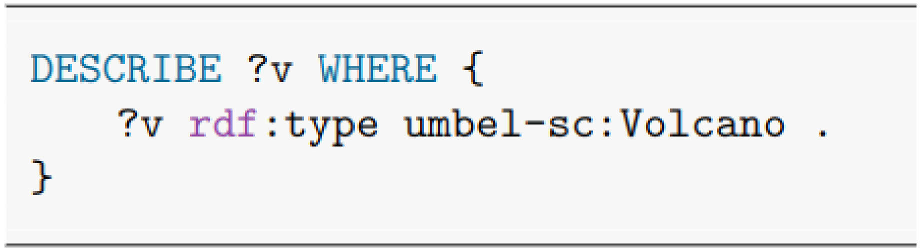
SELECT Query

- Selects a sequence of solutions
- Given a set of constraints in form of
  - Triple patterns
  - Filters
  - Expressions

```
SELECT CONCAT(?fName, " ", ?lName) WHERE {
    ?s _:firstName ?fName ;
       _:lastName ?lName ;
       _:age ?age .
    FILTER(?age > 50)
}
```

DESCRIBE Query

- Returns subgraph describing provided set of terms
- Documentation is underspecified on ”How"
  - (Default) Direct unidirectional/bidirectional connections
  - also Concise Bounded Description (CBD) [1]

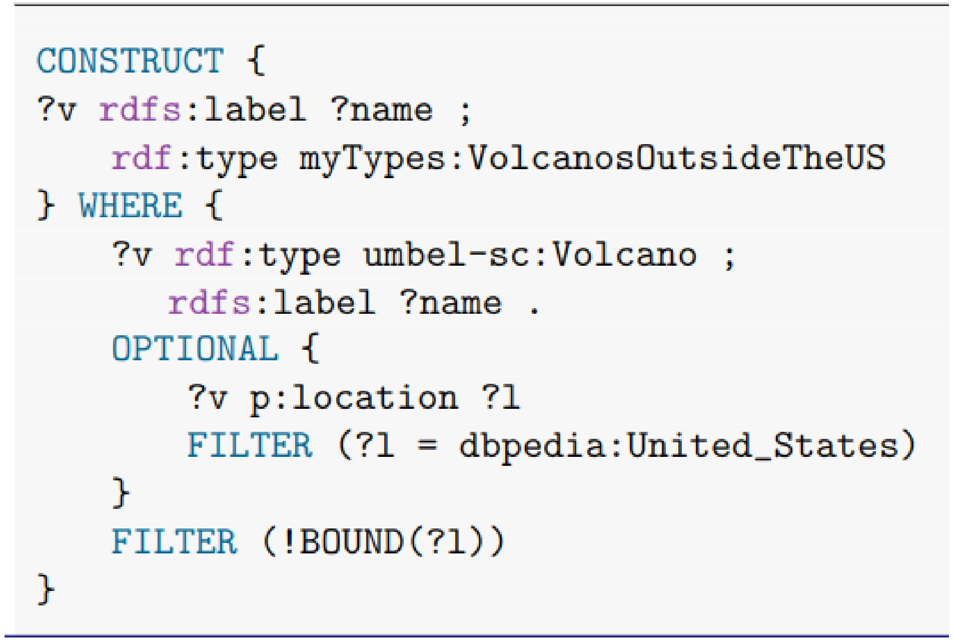


```
DESCRIBE ?v WHERE {
    ?v rdf:type umbel-sc:Volcano .
}
```

[1] https://www.w3.org/Submission/CBD

CONSTRUCT Query

- Returns an RDF graph constructed from provided template
- Note that domains for properties must match

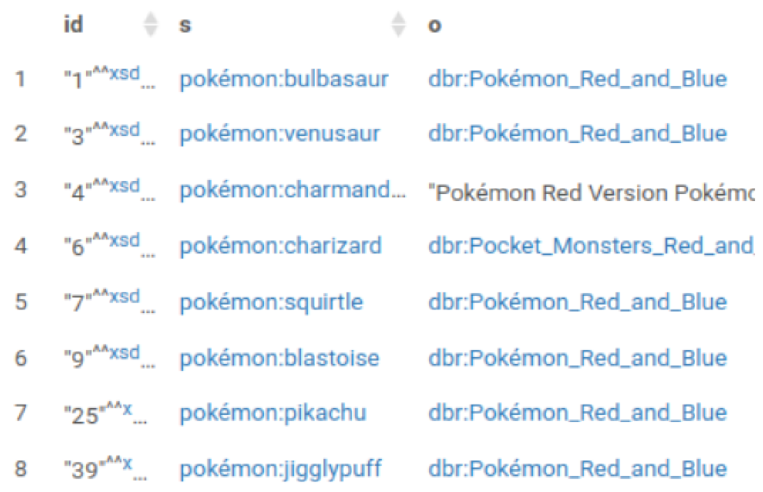


```
CONSTRUCT {
?v rdfs:label ?name ;
   rdf:type myTypes:VolcanosOutsideTheUS
} WHERE {
   ?v rdf:type umbel-sc:Volcano ;
      rdfs:label ?name .
   OPTIONAL {
      ?v p:location ?l
      FILTER (?l = dbpedia:United_States)
   }
   FILTER (!BOUND(?l))
}
```

Services

- SERVICE keyword invokes non-standard functionality given a service resource
- Most common usage is ”Federated Queries"
  - Run subqueries on a different endpoint

```
SELECT ?id ?s ?o WHERE {
  ?s owl:sameAs ?alias .
  ?s pokemon:nationalNumber ?id .

  SERVICE <https://api.triplydb.com/../dbpedia/sparql> {
    ?alias dbp:firstgame ?o
  }
}
```



	id	s	o
1	"1"^^xsd…	pokémon:bulbasaur	dbr:Pokémon_Red_and_Blue
2	"3"^^xsd…	pokémon:venusaur	dbr:Pokémon_Red_and_Blue
3	"4"^^xsd…	pokémon:charmand…	"Pokémon Red Version Pokémo
4	"6"^^xsd…	pokémon:charizard	dbr:Pocket_Monsters_Red_and
5	"7"^^xsd…	pokémon:squirtle	dbr:Pokémon_Red_and_Blue
6	"9"^^xsd…	pokémon:blastoise	dbr:Pokémon_Red_and_Blue
7	"25"^^x…	pokémon:pikachu	dbr:Pokémon_Red_and_Blue
8	"39"^^x…	pokémon:jigglypuff	dbr:Pokémon_Red_and_Blue

Querying Wikidata

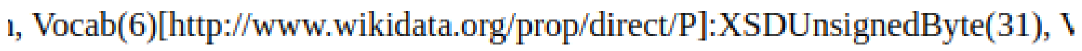
INTRINSICS OF WIKIDATA DATASET DESIGN AND HOW TO USE IT

Wikidata

- 94B triples (53GB compressed and 149GB uncompressed)
- Has public SPARQL endpoint
  - runs Blazegraph
  - has 60sec timeout
- Not their primary data store
  - Dataset is updated from their internal database

Redefining standards

- Redefines RDF standards for performance reasons
- IRIs are illegible
  - rdfs:subClassOf becomes wdt:P279
  - rdf:type becomes wdt:P31
  - Use search api or public examples



```
ı, Vocab(6)[http://www.wikidata.org/prop/direct/P]:XSDUnsignedByte(31), V
```

Reification

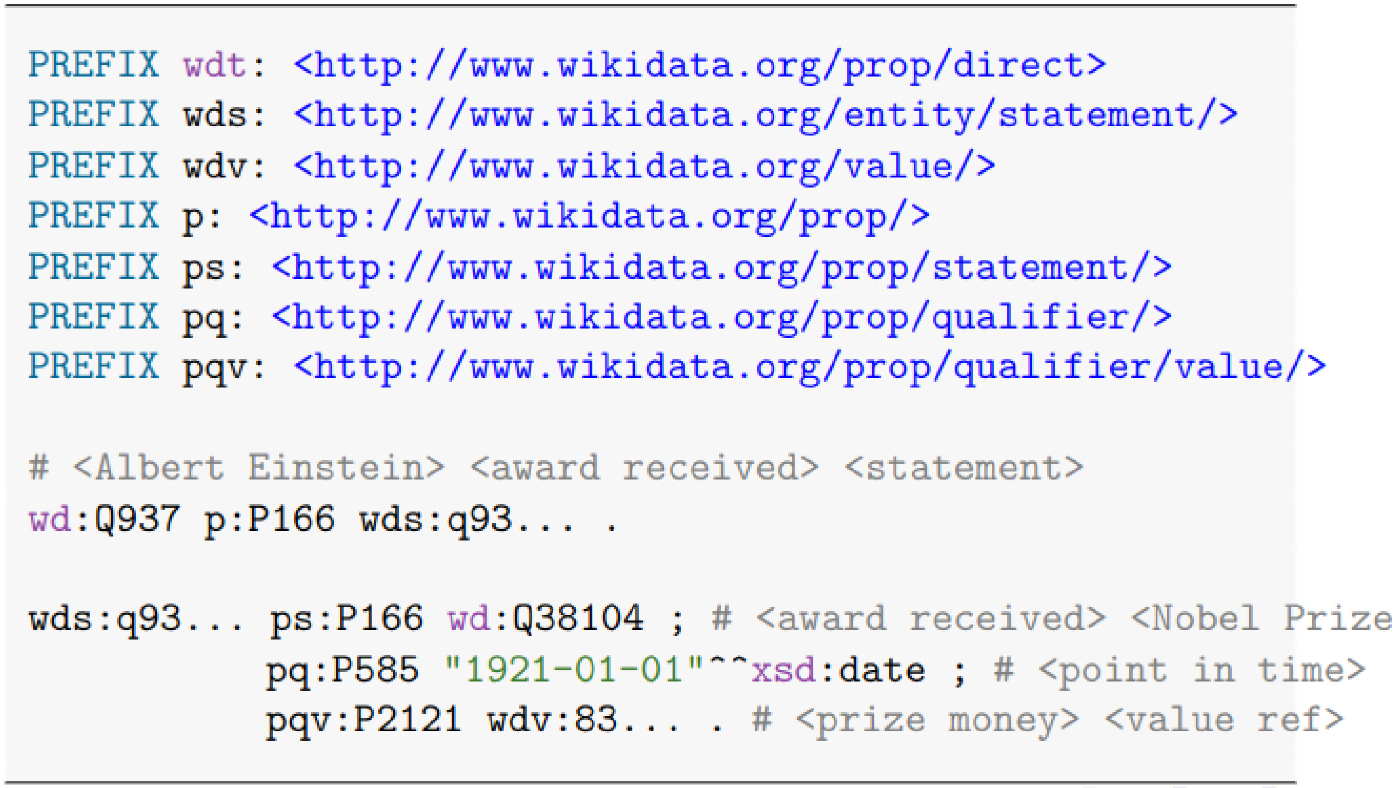
- Facts are expressed as statements: (subject, predicate, object)
  - ”Albert Einstein won Nobel Prize in Physics”
  - _:AlbertEinstein _:won _:NobelPrizeInPhysics
- Sometimes not enough to convey the information:
  - "Albert Einstein won Nobel Prize in Physics in 1921"
  - **Q: how would you represent this in RDF?**

Reification

- Facts are expressed as statements: (subject, predicate, object)
  - ”Albert Einstein won Nobel Prize in Physics”
  - _:AlbertEinstein _:won _:NobelPrizeInPhysics
- Sometimes not enough to convey the information:
  - "Albert Einstein won Nobel Prize in Physics in 1921"
  - **Q: how would you represent this in RDF?**

```
_:triple1 rdf:type rdf:Statement ;
           rdf:subject _:AlbertEinstein ;
           rdf:predicate _:won ;
           rdf:object _:NobelPrizeInPhysics ;
           _:in "1921"^^xsd:date
```

Reification in Wikidata

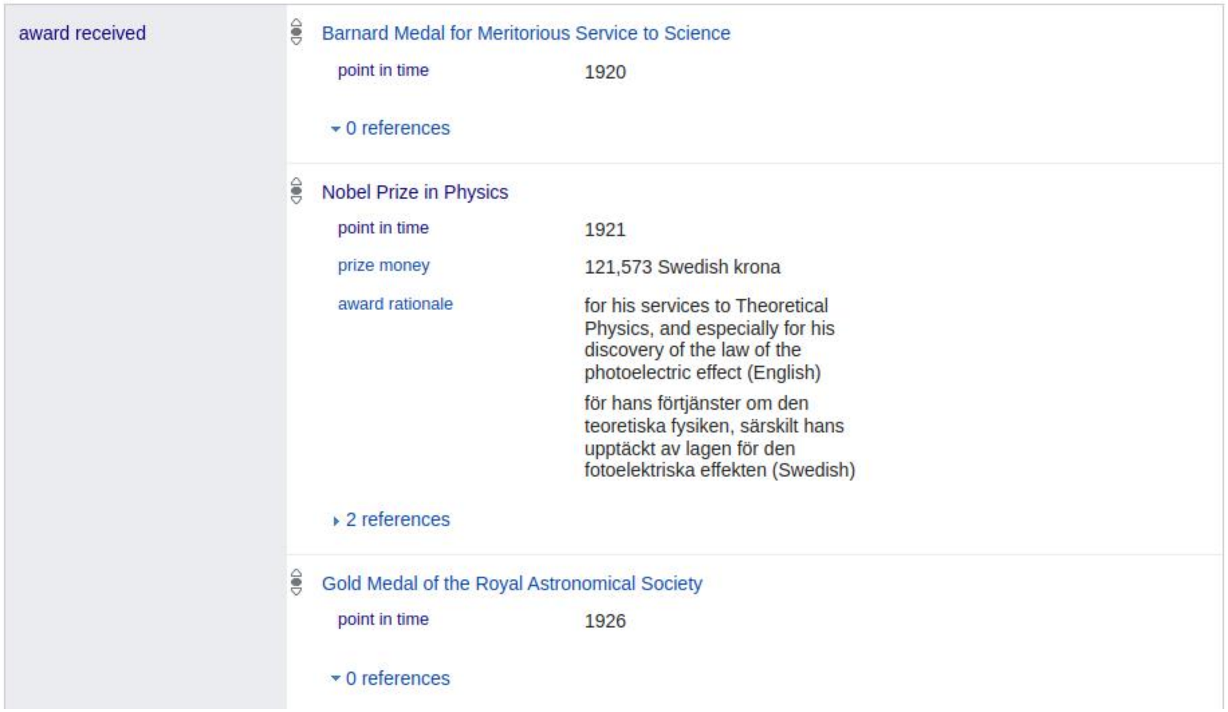


```
PREFIX wdt: <http://www.wikidata.org/prop/direct>
PREFIX wds: <http://www.wikidata.org/entity/statement/>
PREFIX wdv: <http://www.wikidata.org/value/>
PREFIX p: <http://www.wikidata.org/prop/>
PREFIX ps: <http://www.wikidata.org/prop/statement/>
PREFIX pq: <http://www.wikidata.org/prop/qualifier/>
PREFIX pqv: <http://www.wikidata.org/prop/qualifier/value/>

# <Albert Einstein> <award received> <statement>
wd:Q937 p:P166 wds:q93... .

wds:q93... ps:P166 wd:Q38104 ; # <award received> <Nobel Prize
           pq:P585 "1921-01-01"^^xsd:date ; # <point in time>
           pqv:P2121 wdv:83... . # <prize money> <value ref>
```

For full overview: https://www.mediawiki.org/wiki/Wikibase/Indexing/RDF_Dump_Format#Prefixes_used

Reification in Wikidata

- When can you use reified statements?



award received	Barnard Medal for Meritorious Service to Science	
	point in time	1920
	0 references	
	Nobel Prize in Physics	
	point in time	1921
	prize money	121,573 Swedish krona
	award rationale	for his services to Theoretical Physics, and especially for his discovery of the law of the photoelectric effect (English)
		för hans förtjänster om den teoretiska fysiken, särskilt hans upptäckt av lagen för den fotoelektriska effekten (Swedish)
	2 references	
	Gold Medal of the Royal Astronomical Society	
	point in time	1926
	0 references	

Labeling Service

- Adds label selection for variables suffixed with ”Label"

```
SELECT ?sLabel ?p ?oLabel {
    ?s ?p ?o .
    SERVICE wikibase:label { bd:serviceParam wikibase:language "en". }
}
```

- Shorthand for

```
OPTIONAL {
    ?o rdfs:label ?oLabel .
    FILTER (lang(?oLabel) = 'en')
  }
```

- Is anecdotally faster and cleaner

Labeling Service

- Adds label selection for variables suffixed with ”Label

```
SELECT ?sLabel ?p ?oLabel {
    ?s ?p ?o .
    SERVICE wikibase:label { bd:serviceParam wikibase:language "en". }
}
```

- Labels are not defined for direct properties
  - Must use reified predicate variant

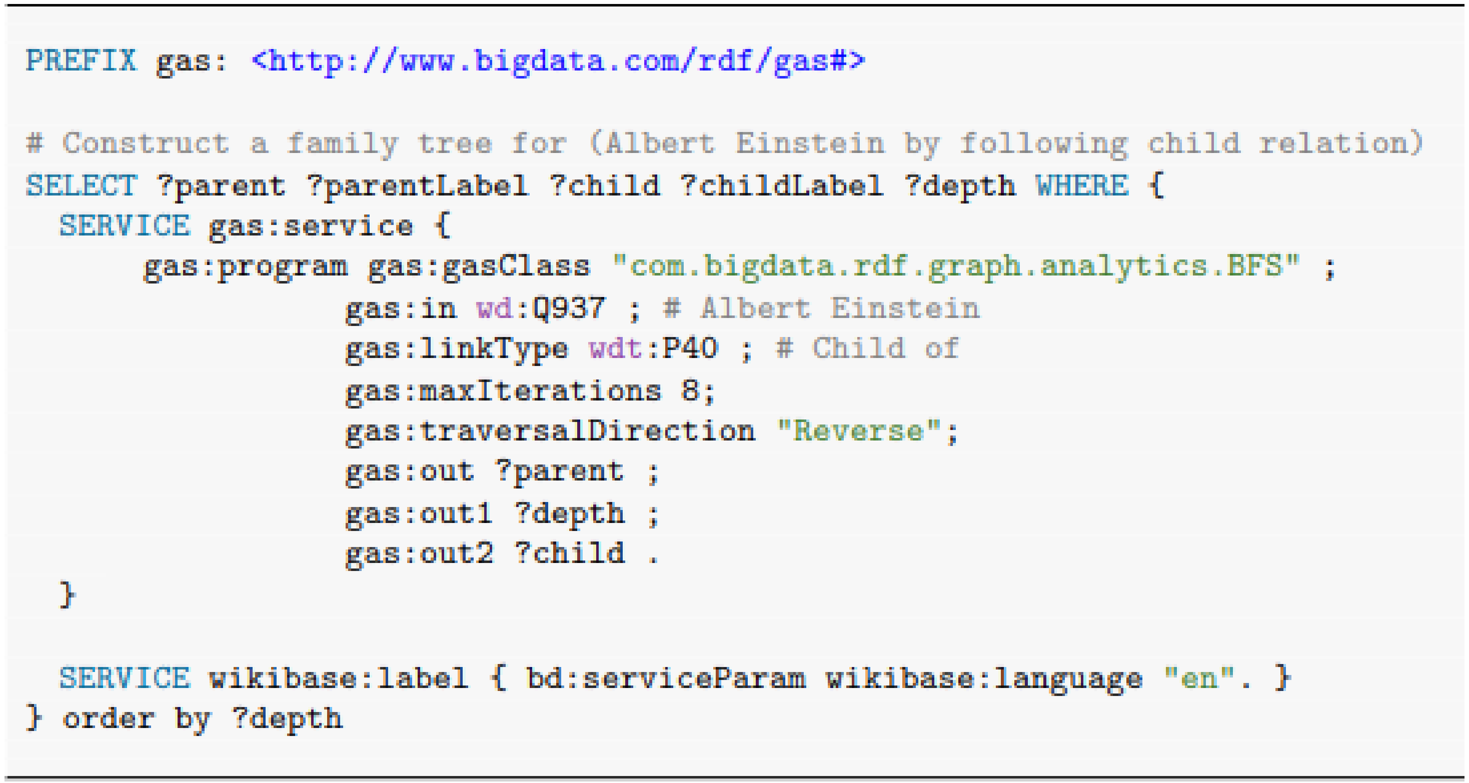
```
SELECT ?s ?pLabel ?o {
    ?s ?pDirect ?o .
    ?p wikibase:directClaim ?pDirect .
    SERVICE wikibase:label { bd:serviceParam wikibase:language "en". }
}
```

Gather Apply Scatter (GAS) Service

- Blazegraph API for implementing graph algorithms [1]
- Various implementations are available: Breadth First Search, Single Source Shortest Path, Connected Component and Page Rank
- Generally faster than equivalent CONSTRUCT queries

[1] https://github.com/blazegraph/database/wiki/RDF_GAS_API

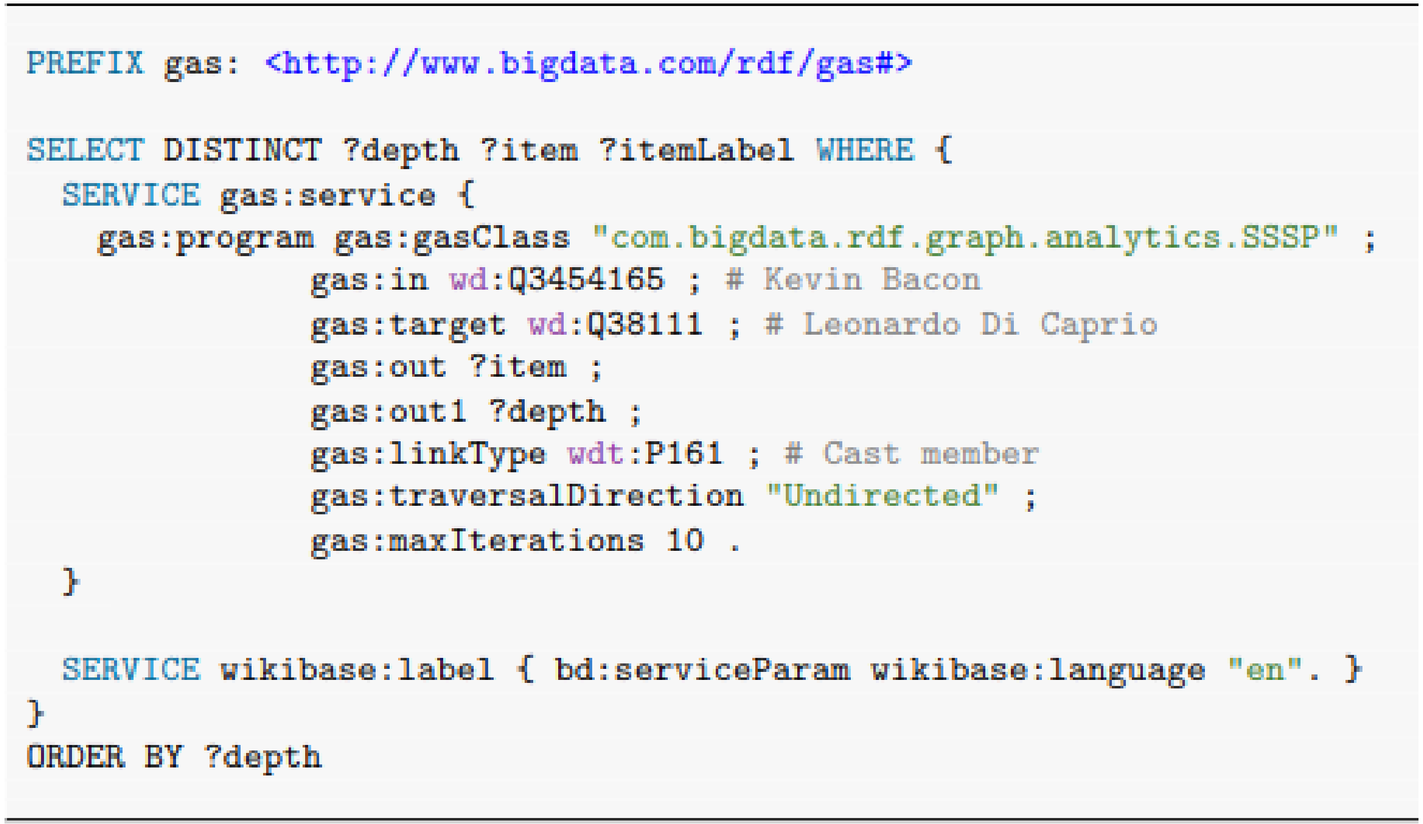
GAS Service: BFS



```
PREFIX gas: <http://www.bigdata.com/rdf/gas#>

# Construct a family tree for (Albert Einstein by following child relation)
SELECT ?parent ?parentLabel ?child ?childLabel ?depth WHERE {
  SERVICE gas:service {
       gas:program gas:gasClass "com.bigdata.rdf.graph.analytics.BFS" ;
                   gas:in wd:Q937 ; # Albert Einstein
                   gas:linkType wdt:P40 ; # Child of
                   gas:maxIterations 8;
                   gas:traversalDirection "Reverse";
                   gas:out ?parent ;
                   gas:out1 ?depth ;
                   gas:out2 ?child .
  }

  SERVICE wikibase:label { bd:serviceParam wikibase:language "en". }
} order by ?depth
```

Try It!

GAS Service: Single Source Shortest Path



```
PREFIX gas: <http://www.bigdata.com/rdf/gas#>

SELECT DISTINCT ?depth ?item ?itemLabel WHERE {
  SERVICE gas:service {
    gas:program gas:gasClass "com.bigdata.rdf.graph.analytics.SSSP" ;
                gas:in wd:Q3454165 ; # Kevin Bacon
                gas:target wd:Q38111 ; # Leonardo Di Caprio
                gas:out ?item ;
                gas:out1 ?depth ;
                gas:linkType wdt:P161 ; # Cast member
                gas:traversalDirection "Undirected" ;
                gas:maxIterations 10 .
  }

  SERVICE wikibase:label { bd:serviceParam wikibase:language "en". }
}
ORDER BY ?depth
```

Try It!

Conclusion: Wikidata

- Wikidata is too large, use all the tools you have
- Use the query service:
  - https://query.wikidata.org/
- Example queries:
  - https://www.wikidata.org/wiki/Wikidata:SPARQL_query_service/queries/examples

Query Performance

TIPS AND TRICKS TO SPEED UP YOUR QUERIES

Query Performance 101

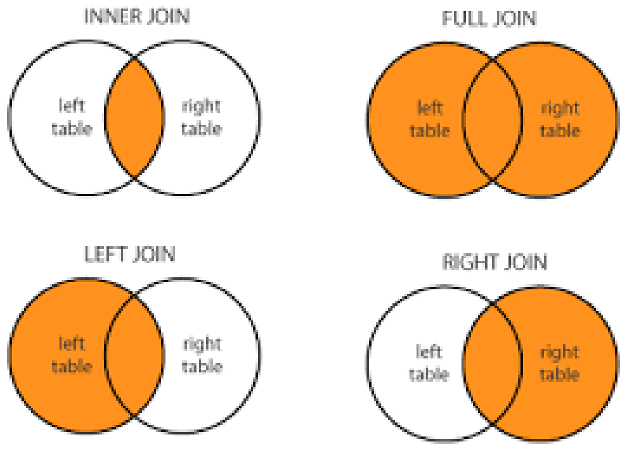
- Every triple pattern introduces a join
- Common sense applies
  - Reduce the joins
  - Visit fewer triples (add more constraints)
- Analytical queries often include ORDER BY, DISTINCT, and subquery requiring visiting and keeping all triples
  - Referred to as full materialization
  - Requires a lot of memory

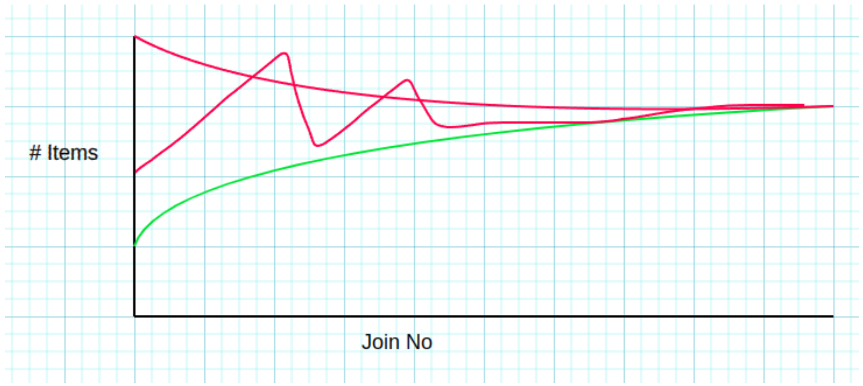
Query Optimizer

- Responsibilities:
  - Narrow down number of triples accessed
  - Pick favorable indexes
  - Reduce join depth
  - Minimize materialization
- Not as smart as Relational DB query optimizer
  - Heuristics are less reliable when working with a single table
  - All data is stored as a string
  - Unused bindings are harder to detect

Join Ordering

- Table cardinality
  - Number of items stored in a table
- Join ratio
  - Proportion of rows before to rows after the join
- Joining in an increasing order results in less work
  - Most of the joins are inner joins
  - OPTIONAL is left join

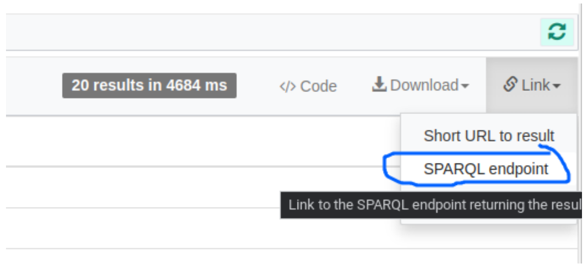




[1] Joins: https://www2.stat.duke.edu/~cr173/Sta323_Sp19/slides/Lec16.html#29

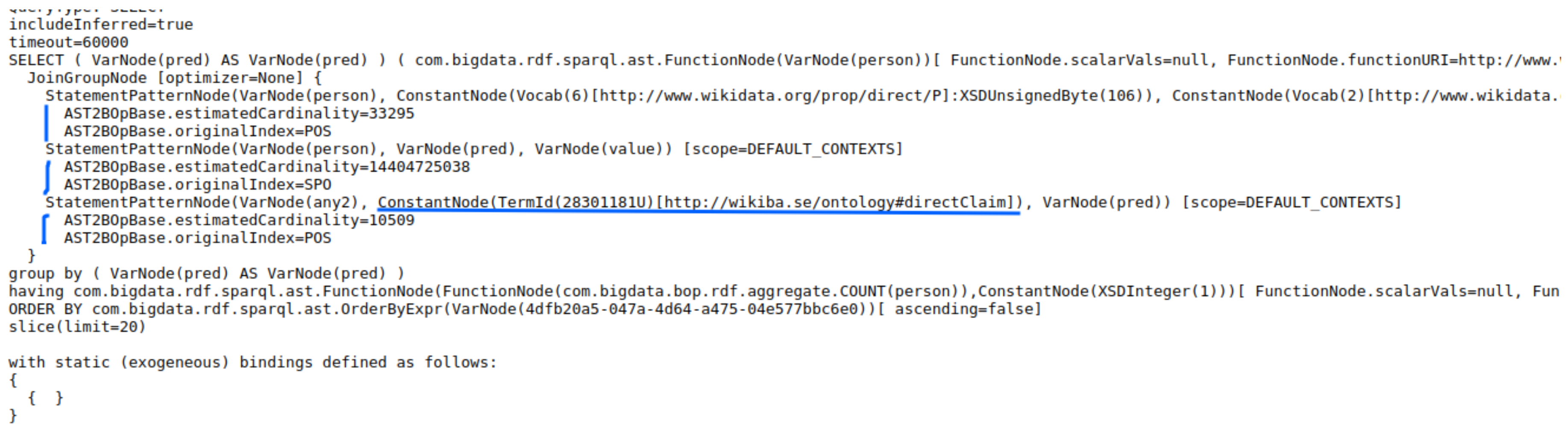
When the Query Optimizer fails

- Sometimes the query looks sane but times out
  - Query might be fault (use of unbound variables)
  - Query does too much
  - Query optimizer creates a suboptimal plan
- Use query "explain" output to check how it is executed
- In wikidata:
  - Copy sparql endpoint url
  - Add "&explain" at the end
  - Open url in browser
  - Try it!



What does explain show?

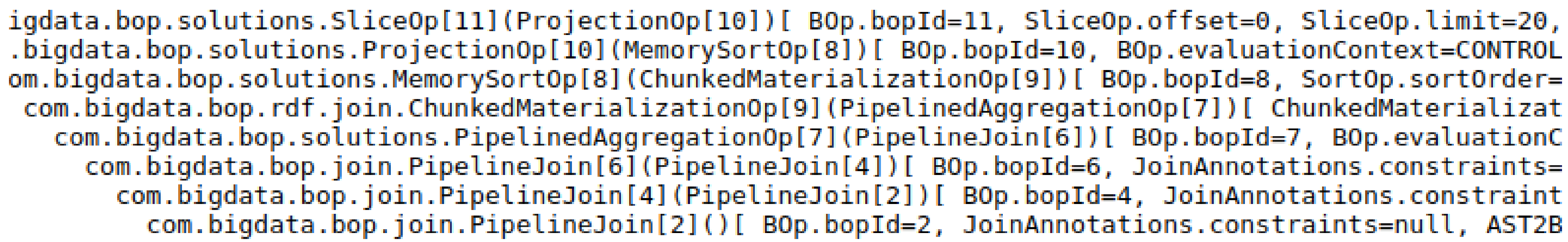
- Optimized AST
  - Shows the semantics of how your query will be run
  - Rationale behind join reorders (estimated cardinality)
  - Shows the indexes used



```
includeInferred=true
timeout=60000
SELECT ( VarNode(pred) AS VarNode(pred) ) ( com.bigdata.rdf.sparql.ast.FunctionNode(VarNode(person))[ FunctionNode.scalarVals=null, FunctionNode.functionURI=http://www.
  JoinGroupNode [optimizer=None] {
    StatementPatternNode(VarNode(person), ConstantNode(Vocab(6)[http://www.wikidata.org/prop/direct/P]:XSDUnsignedByte(106)), ConstantNode(Vocab(2)[http://www.wikidata.
      AST2BOpBase.estimatedCardinality=33295
      AST2BOpBase.originalIndex=POS
    StatementPatternNode(VarNode(person), VarNode(pred), VarNode(value)) [scope=DEFAULT_CONTEXTS]
      AST2BOpBase.estimatedCardinality=14404725038
      AST2BOpBase.originalIndex=SPO
    StatementPatternNode(VarNode(any2), ConstantNode(TermId(28301181U)[http://wikiba.se/ontology#directClaim]), VarNode(pred)) [scope=DEFAULT_CONTEXTS]
      AST2BOpBase.estimatedCardinality=10509
      AST2BOpBase.originalIndex=POS
  }
group by ( VarNode(pred) AS VarNode(pred) )
having com.bigdata.rdf.sparql.ast.FunctionNode(FunctionNode(com.bigdata.bop.rdf.aggregate.COUNT(person)),ConstantNode(XSDInteger(1)))[ FunctionNode.scalarVals=null, Fun
ORDER BY com.bigdata.rdf.sparql.ast.OrderByExpr(VarNode(4dfb20a5-047a-4d64-a475-04e577bbc6e0))[ ascending=false]
slice(limit=20)

with static (exogeneous) bindings defined as follows:
{
  {  }
}
```

What does explain show?

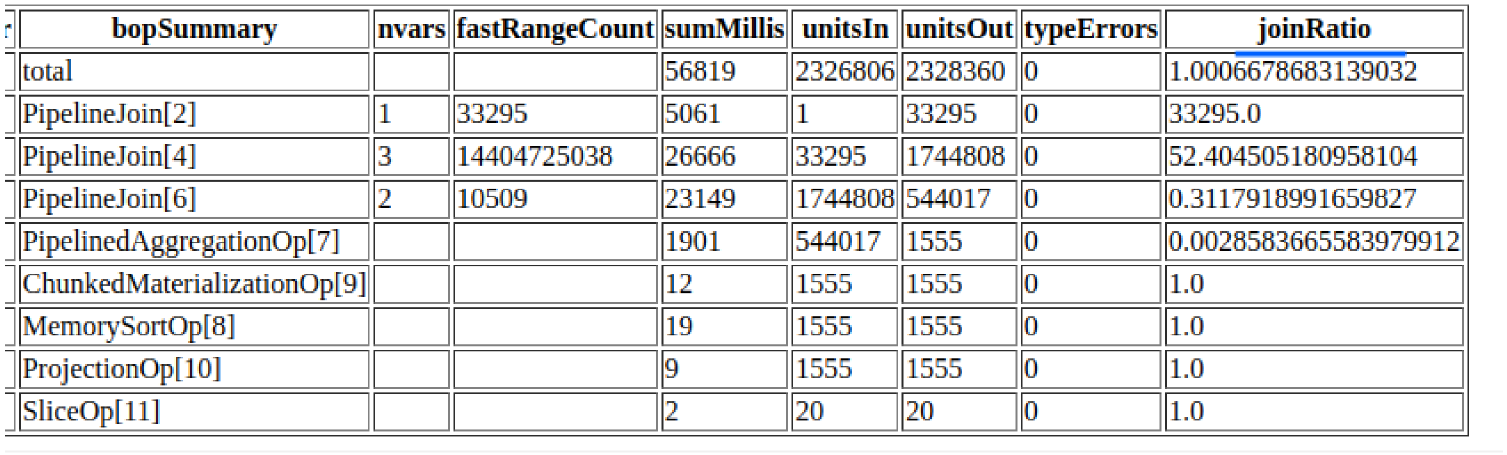
- Query Plan
  - Shows the exact operations executed within the query pipeline



```
igdata.bop.solutions.SliceOp[11](ProjectionOp[10])[ BOp.bopId=11, SliceOp.offset=0, SliceOp.limit=20,
.bigdata.bop.solutions.ProjectionOp[10](MemorySortOp[8])[ BOp.bopId=10, BOp.evaluationContext=CONTROL
om.bigdata.bop.solutions.MemorySortOp[8](ChunkedMaterializationOp[9])[ BOp.bopId=8, SortOp.sortOrder=
 com.bigdata.bop.rdf.join.ChunkedMaterializationOp[9](PipelinedAggregationOp[7])[ ChunkedMaterializat
   com.bigdata.bop.solutions.PipelinedAggregationOp[7](PipelineJoin[6])[ BOp.bopId=7, BOp.evaluationC
     com.bigdata.bop.join.PipelineJoin[6](PipelineJoin[4])[ BOp.bopId=6, JoinAnnotations.constraints=
       com.bigdata.bop.join.PipelineJoin[4](PipelineJoin[2])[ BOp.bopId=4, JoinAnnotations.constraint
         com.bigdata.bop.join.PipelineJoin[2]()[ BOp.bopId=2, JoinAnnotations.constraints=null, AST2B
```

What does explain show?

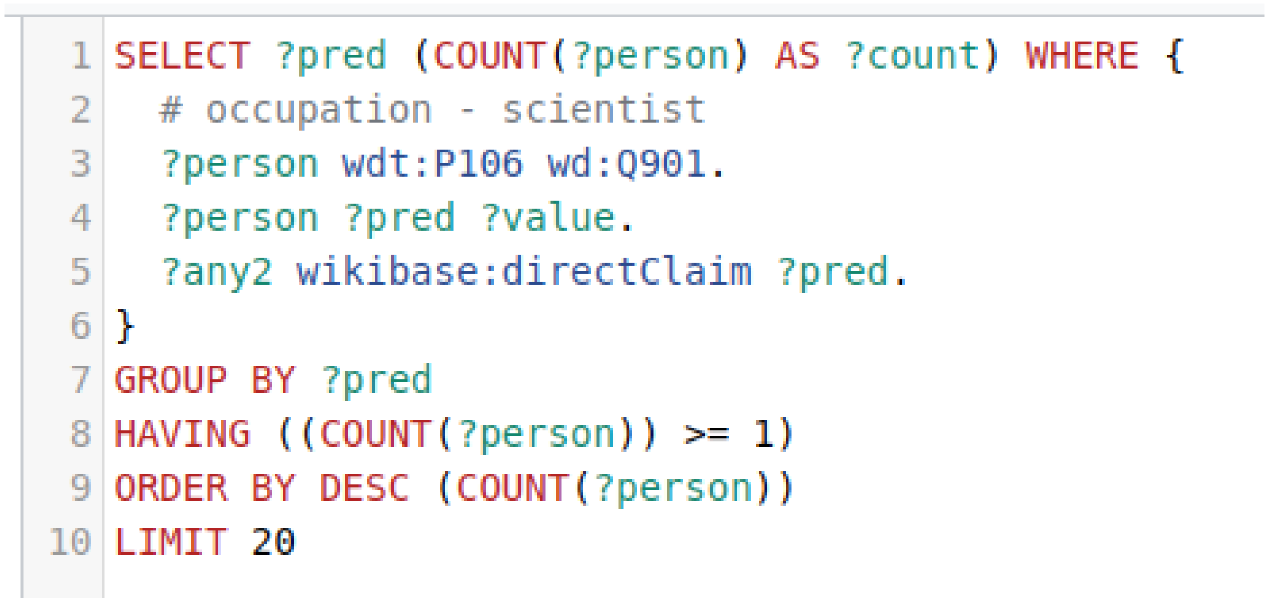
- Query Evaluation Statistics
  - Is computed after executions
  - Times each operation
  - Shows join ratio of each join



bopSummary	nvars	fastRangeCount	sumMillis	unitsIn	unitsOut	typeErrors	joinRatio
total			56819	2326806	2328360	0	1.0006678683139032
PipelineJoin[2]	1	33295	5061	1	33295	0	33295.0
PipelineJoin[4]	3	14404725038	26666	33295	1744808	0	52.404505180958104
PipelineJoin[6]	2	10509	23149	1744808	544017	0	0.3117918991659827
PipelinedAggregationOp[7]			1901	544017	1555	0	0.0028583665583979912
ChunkedMaterializationOp[9]			12	1555	1555	0	1.0
MemorySortOp[8]			19	1555	1555	0	1.0
ProjectionOp[10]			9	1555	1555	0	1.0
SliceOp[11]			2	20	20	0	1.0

Demo 1: What is wrong with this query?

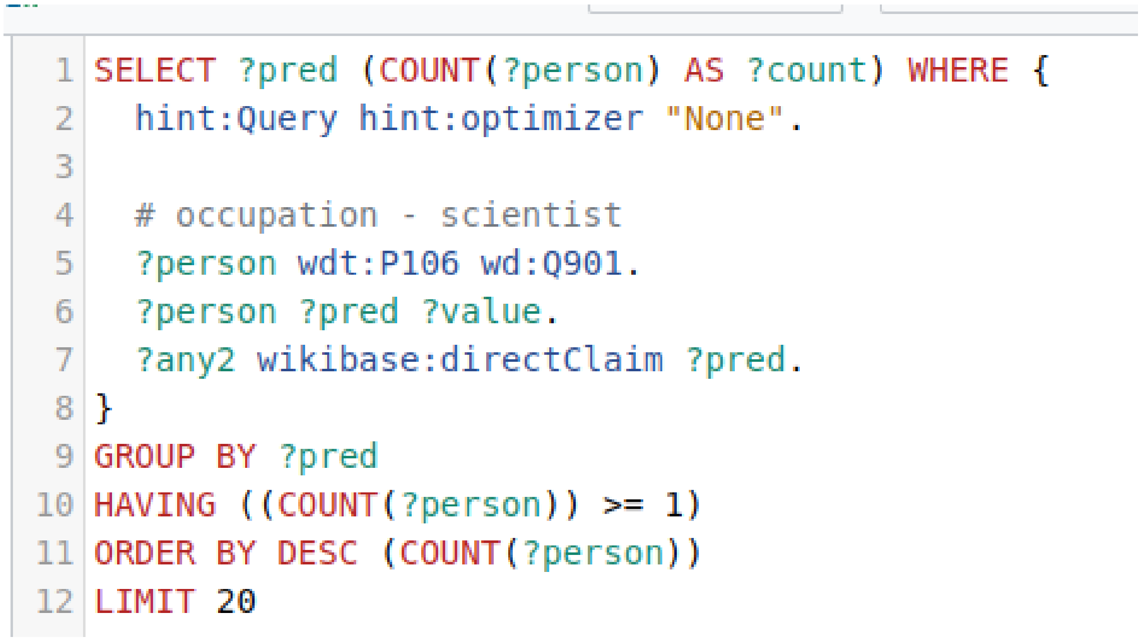
This query times out



```
SELECT ?pred (COUNT(?person) AS ?count) WHERE {
  # occupation - scientist
  ?person wdt:P106 wd:Q901.
  ?person ?pred ?value.
  ?any2 wikibase:directClaim ?pred.
}
GROUP BY ?pred
HAVING ((COUNT(?person)) >= 1)
ORDER BY DESC (COUNT(?person))
LIMIT 20
```

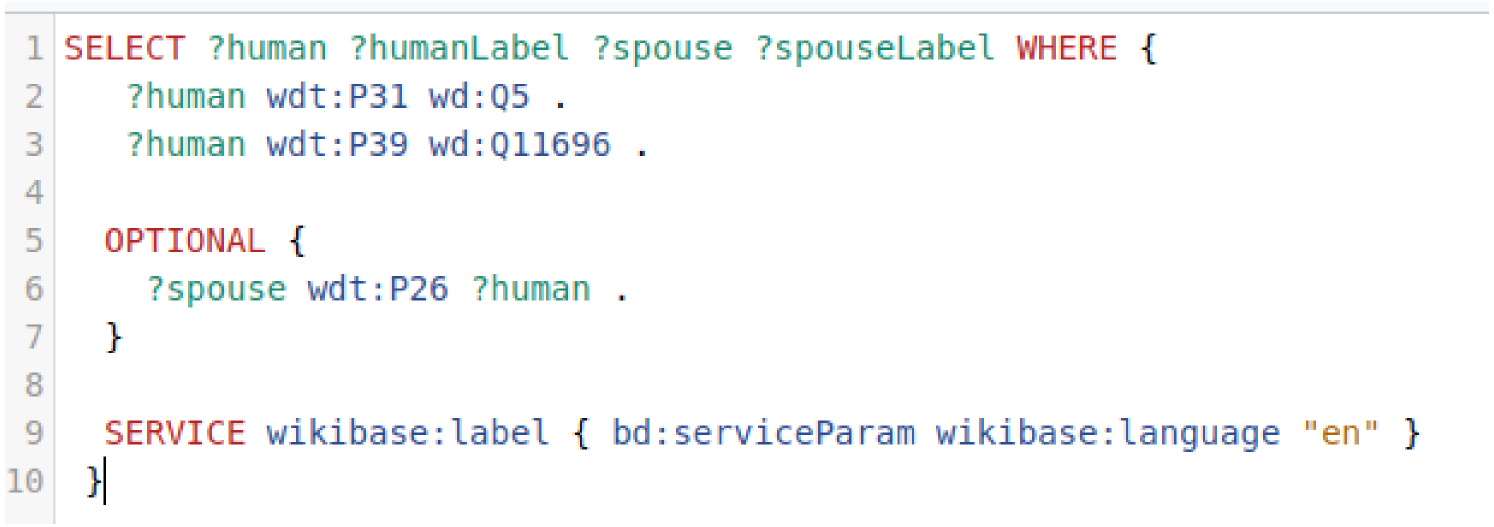
Demo 1: What is wrong with this query?

Joins are in correct order, but query optimizer reorders them.
DirectClaim is used as first join but since second join is unconstrained it selects the whole dataset for the next join



```
SELECT ?pred (COUNT(?person) AS ?count) WHERE {
  hint:Query hint:optimizer "None".

  # occupation - scientist
  ?person wdt:P106 wd:Q901.
  ?person ?pred ?value.
  ?any2 wikibase:directClaim ?pred.
}
GROUP BY ?pred
HAVING ((COUNT(?person)) >= 1)
ORDER BY DESC (COUNT(?person))
LIMIT 20
```

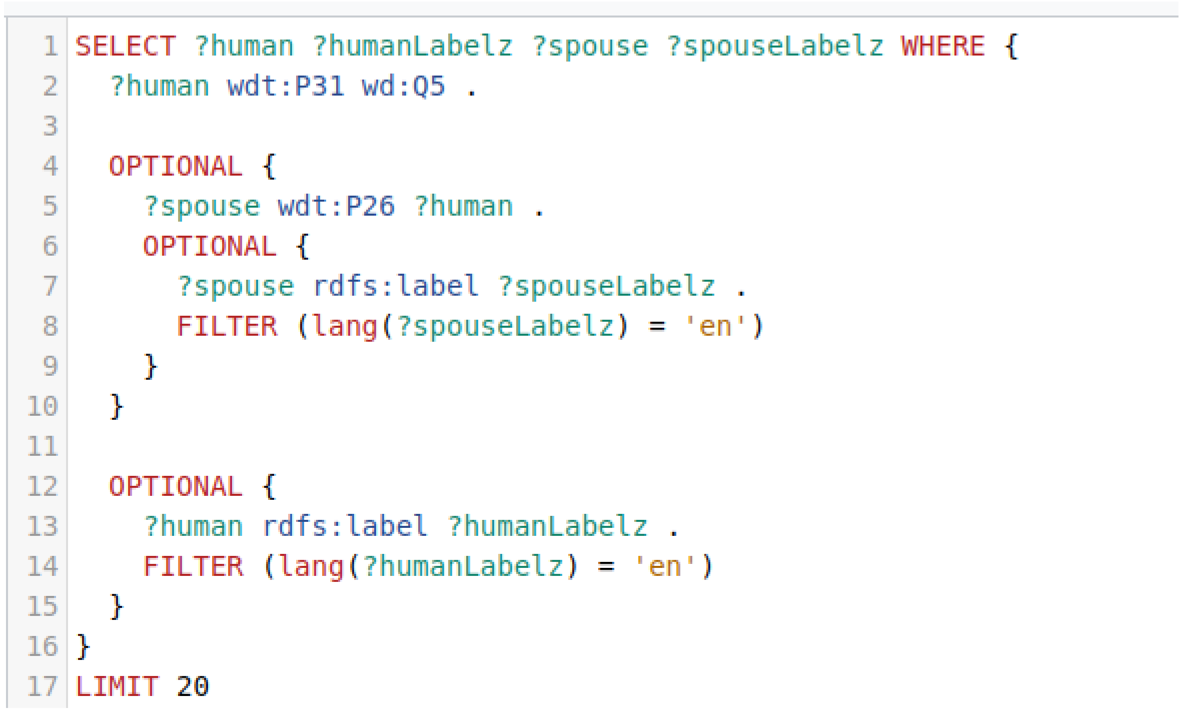
Demo 2: What is wrong with this query?



```
SELECT ?human ?humanLabel ?spouse ?spouseLabel WHERE {
   ?human wdt:P31 wd:Q5 .
   ?human wdt:P39 wd:Q11696 .

  OPTIONAL {
    ?spouse wdt:P26 ?human .
  }

  SERVICE wikibase:label { bd:serviceParam wikibase:language "en" }
 }
```

Demo 2: What is wrong with this query?

Service adds joins at root level.
Because spouse is optional this results in a lot of extra work trying to join on an unbound value



```
SELECT ?human ?humanLabelz ?spouse ?spouseLabelz WHERE {
  ?human wdt:P31 wd:Q5 .

  OPTIONAL {
    ?spouse wdt:P26 ?human .
    OPTIONAL {
      ?spouse rdfs:label ?spouseLabelz .
      FILTER (lang(?spouseLabelz) = 'en')
    }
  }

  OPTIONAL {
    ?human rdfs:label ?humanLabelz .
    FILTER (lang(?humanLabelz) = 'en')
  }
}
LIMIT 20
```

Conclusion

- Introduction to BOLD
  - Feel free to use the code or demo to explore
  - Start with TriplyDB
- Triple Stores
  - Pick the triple store you need
- SPARQL
  - There are many right things to do something
- Querying Wikidata
  - Wikidata is unique for a reason
- Query Performance
  - Sometimes query optimizer is dumb. Work around it