universität freiburg

Databases and Information Systems WS 25/26

Lecture 7: Knowledge Graphs, SPARQL, SPARQL to SQL

December 2, 2025

Prof. Dr. Hannah Bast
Department of Computer Science
University of Freiburg

Overview of this lecture

- Organizational
  - Your experiences with ES6 — query optimization
- Content
  - Knowledge graphs / RDF — all data is modelled as triples
  - SPARQL — the query language for RDF
  - Reification — complex information as triples
  - SPARQL to SQL — a generic translation algorithm

ES7: Implement the SPARQL to SQL translation and use it to process SPARQL queries on two datasets

Experiences with ES6 1/2

- Excerpts from your feedback [somewhat representative]

  "Nice exercises, I got a way deeper understanding of the algorithm"

  "Exercise 1 was fun and helped a lot to understand the algorithm"

  "Exercise 1 really helped understanding how to implement Ex .2"

  "Exercise 1 took more time than expected but it reinforced the understanding of the algorithm and showed some special cases"

  "Easiest sheet so far; Exercise 2 was straightforward and the only annoying thing was computing Exercise 1 by hand"

  "Took longer to get the frozensets right than the actual task"

  "Explanation of the algor. by an example would have been helpful"

  "The greatest painpoint is that you are not using git"

Experiences with ES6 2/2

- How did you like relational databases?

  "I loved that we implemented our own SELECT, WHERE, ... and I feel like now understand what happens in the background"

  "Overall I am quite happy with the topic and the selection, except with this exercise sheet [I would have liked to go deeper]"

  "Exciting to implement our own database engine along the way"

  "Was really helpful that throughout the lectures we always worked with the same real data + the imdb/movies theme was cool"

  "I learned many useful things about this [new for me] topic"

  "ES 6: I would have preferred to do the initial implementation myself … better find a different representation for the lecture"

  [maybe next time: L6 → show examples, ES6 → implement it]

Knowledge Graphs 1/7

- Resource Description Framework
  - RDF is based on a strikingly simple principle: ALL DATA, whatever it is, is modelled as a **set of triples**

    subject predicate object .

  - For example:

<Nicole_Kidman>	<acted_in>	<Eyes_Wide_Shut>	.
<Brad_Pitt>	<acted_in>	<Burn_After_Reading>	.
<Joel_Coen>	<directed>	<Burn_After_Reading>	.
<Ethan_Coen>	<directed>	<Burn_After_Reading>	.
<Tom_Cruise>	<acted_in>	<Eyes_Wide_Shut>	.
<Brad_Pitt>	<acted_in>	<Eyes_Wide_Shut>	.
<Nicole_Kidman>	<married_to>	<Tom_Cruise>	.
<Stanley Kubrick>	<directed>	<Eyes_Wide_Shut>	.

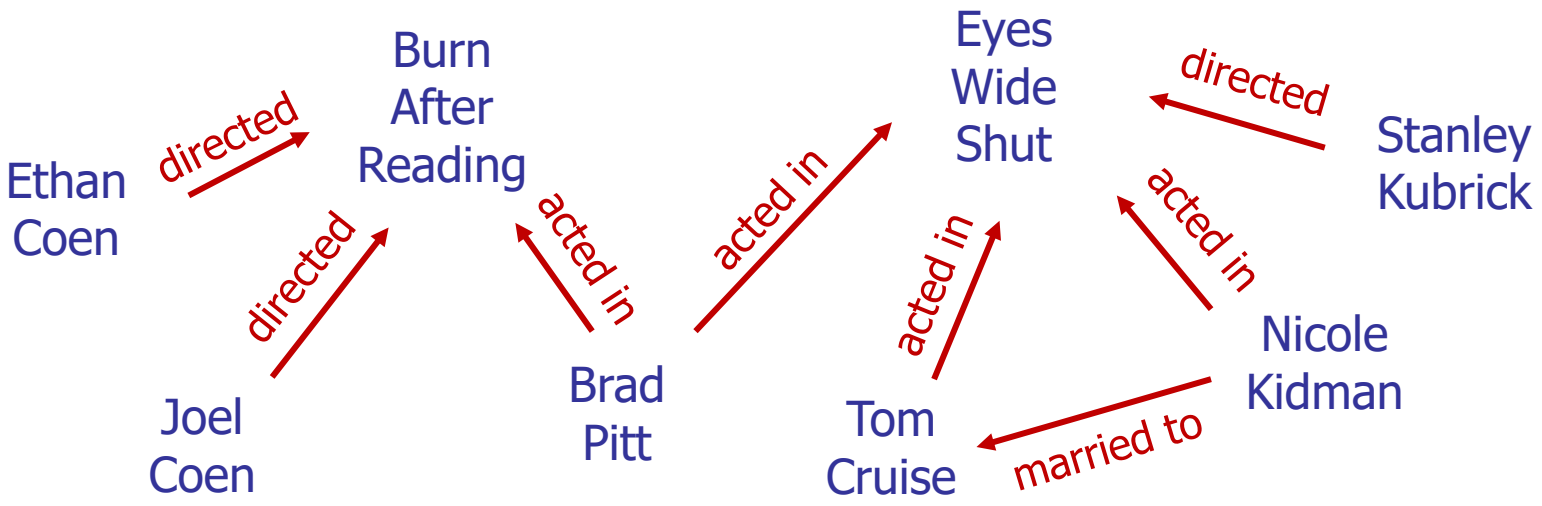
Wrong?
(but it's a triple)

Knowledge Graphs 2/7

- Graph representation
  - The triples of a KG form a **directed graph** as follows:

    Nodes: Each distinct entity is one node

    Edges: Each triple is a directed edge from the subject node to the object node



Knowledge Graphs 3/7

wdt:P19 ~ place of birth
wd:Q18094 ~ Honolulu
wdt:P569 ~ date of birth

- International Resource Identifier (IRI)
  - Names have globally unique identifiers, so-called IRIs

    An IRI is like an address in a browser (URL), only that Unicode characters like ä, ö, ü, … are allowed
  - The IRI is often not human-readable but ID-like; instead there are dedicated triples for its names and aliases
  - For example, the Wikidata IRI for Nicole Kidman is

    https://www.wikidata.org/entity/Q37459

    and some example triples (with **IRI prefixes**) look like this:

    wd:Q37459 **rdfs:**label "निकोल किडमैन"@hi
    wd:Q37459 **wdt:**P19 **wd:**Q18094
    wd:Q37459 **wdt:**P569 "1967-06-20"^^xsd:date

Knowledge Graphs 4/7

- Some widely used knowledge graphs
  - **Wikidata:** Started 2013, successor of Freebase (bought by Google in 2010 for 99M$), crowd-sourced, amazing coverage

    21 **B** triples, 62 **k** predicates
  - **UniProt:** Started 2002, protein sequences, genes, all kinds of metadata, …

    240 **B** triples, 312 predicates
  - **PubChem:** Started 2004, chemical compounds and substances and their effects on organisms

    26 **B** triples, 473 predicates

Knowledge Graphs 5/7

- And more …
  - **OpenStreetMap:** Started 2004, all the geo-data of the world, crowd-sourced, amazing coverage and quality

    202 **B** triples, 106 **k** predicates

  - **DBLP:** Started 1993, publication meta-data for computer science and adjacent fields

    1.5 **B** triples, 96 predicates

  - Most of the IT-heavy companies maintain their own (very large) knowledge graph:

    Google, Amazon, Microsoft, Meta, Bloomberg, Walmart, Airbnb, …

Knowledge Graphs 6/7

- The standard query language for RDF is
  - SPARQL is a variant of SQL, adapted to the (simple) RDF
  - For example, the following query searches Wikidata for the names of all people born in Honolulu (see triples on slide 6)

```
PREFIX wd: <http://www.wikidata.org/entity/>
PREFIX wdt: <http://www.wikidata.org/prop/direct/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?person ?person_name WHERE {
    ?person wdt:P19 wd:Q18094 .
    ?person rdfs:label ?person_name .
    FILTER (LANG(?person_name) = "en")
}
```

  - More about SPARQL on slides 11 – 16

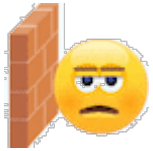
Knowledge Graphs 7/7

This allows "federated" SPARQL queries like

[all power lines in the EU]

- Interoperability
  - One of the strengths of RDF and SPARQL is the great ease, with which different datasets can be **combined**

    For relational databases this can be a hard: different and often complex schemas, different identifiers, etc.



    In RDF, all you need are additional triples relating the IDs from the two datasets you want to combine

    wd:Q183 **wdtn:**P402 **osmrel:**51477 [from Wikidata]

    osmrel:51477 **osm**:wikidata **wd:**Q183 [from OpenStreetMap]

wd:Q183	IRI for Germany in Wikidata
osmrel:51477	IRI for Germany in OpenStreetMap
wdtn:P402	predicate for "OpenStreetMap IRI" in Wikidata
osm:wikidata	predicate for "Wikidata IRI" in OpenStreetMap

SPARQL 1/7

- What is SPARQL
  - The standard query language for RDF data

    SPARQL = **S**PARQL **P**rotocol **A**nd **R**DF **Q**uery **L**anguage
  - In the following, we will learn basic SPARQL by example

    Like SQL, the basic language is simple, but the complete language is quite complex

    Unlike SQL, the standard is freely available on the Web, provided by the W3C: https://www.w3.org/TR/sparql11-query
  - SPARQL and SQL are essentially equivalent

    On slides 27 – 31, we will see how basic SPARQL queries can be translated to basic SQL queries

    It will be Exercise 1 of ES7 to implement this

SPARQL 2/7

- Example query
  - Let us consider the following query on a knowledge graph following the example triples from slide 4

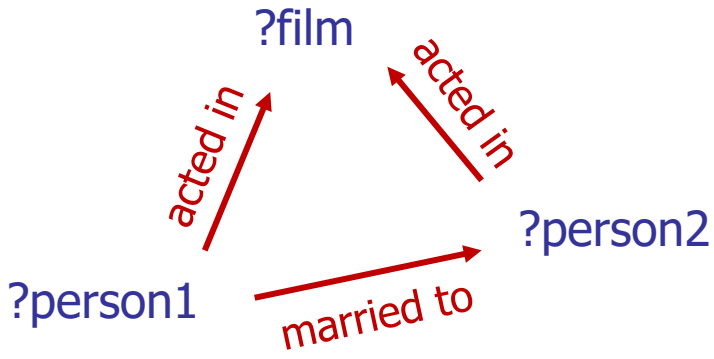
```
SELECT ?person1 ?person2 ?film WHERE {
  ?person1   <acted_in>     ?film  .
  ?person2   <acted_in>     ?film  .
  ?person1   <married_to>   ?person2  .
}

# married couples who acted together in at least one movie
```

  - Like in SQL, the result of a SPARQL query is always a **table**
  - There is one table row for each assignment to the variables such that all the triples in the query body exist

See slide 28 for the equivalent SQL query

SPARQL 3/7

- Viewing SPARQL queries as graph patterns
  - A SPARQL query can be viewed as a sub-graph with variables at some nodes or edges
  - Solving the query then amounts to finding all matches of that (small) sub-graph in the (large) knowledge graph



Each distinct match corresponds to one assignment of values to the variables such that the corresponding sub-graph exists in the knowledge graph

SPARQL 4/7

- PREFIX declarations
  - On "real" knowledge graphs (with long cryptic IRIs) we can abbreviate the IRIs as follows (this is the query from slide 10)

```
PREFIX wd: <http://www.wikidata.org/entity/>
PREFIX wdt: <http://www.wikidata.org/prop/direct/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?person ?person_name WHERE {
  ?person wdt:P19 wd:Q18094 .
  ?person rdfs:label ?person_name .
  FILTER (LANG(?person_name) = "en")
}
```

<http://www.wikidata.org/prop/direct/P19>	place of birth
<http://www.wikidata.org/entity/Q18094>	Honolulu
<http://www.w3.org/2000/01/rdf-schema#label>	canonical name

SPARQL 5/7

- ORDER BY and LIMIT
  - Similar to SQL, except that DESC or ASC is not a suffix (as in SQL) but written like a function of the sort key variable

```
PREFIX wd: <http://www.wikidata.org/entity/>
PREFIX wdt: <http://www.wikidata.org/prop/direct/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?person ?person_name ?date_of_birth WHERE {
   ?person wdt:P19 wd:Q18094 .
   ?person wdt:P569 ?date_of_birth .
   ?person rdfs:label ?person_name .
   FILTER (LANG(?person_name) = "en")
}
ORDER BY DESC(?date_of_birth) LIMIT 100
```

<http://www.wikidata.org/prop/direct/P569> date of birth

SPARQL 6/7

- Incomplete list of differences between SPARQL and SQL
  - Principle differences
    - In SPARQL, we don't need a FROM class (because, by default, all the data is stored in one big table)

      But there is the concept of subgraphs ("named graphs")
    - In SPARQL, the main instrument for formulating a query is a set of triples with variables at some positions

      When the same variables appear in different triples, this corresponds to a JOIN operation, see slides 28 + 30
  - Minor syntactic differences
    - In SPARQL, no comma between variables in SELECT clause
    - In SPARQL, DESC(?var) instead of var DESC

SPARQL 7/7

- A variety of example queries
  - Birth places of people with first name X — Wikidata
  - Notable events that happened on Dec 2 — Wikidata
  - German universities with student count — Wikidata
  - Organisms with their proteins sequences — UniProt
  - NSAID drugs with small molecular weight — PubChem
  - All streets in X — OpenStreetMap
  - Fast food restaurants along the E 35 — Wikidata+OSM

These are all running with the same system, with **almost zero** configuration per dataset, that is the power of RDF+SPARQL

Reification 1/8

- How to model complex information as triples?
  - For example, Meryl Streep won an Oscar for these movies

Kramer vs. Kramer	1980	Best Supporting Actress
Sophie's Choice	1982	Best Actress
The Iron Lady	2011	Best Actress

  - Here is an attempt to model this information as triples:

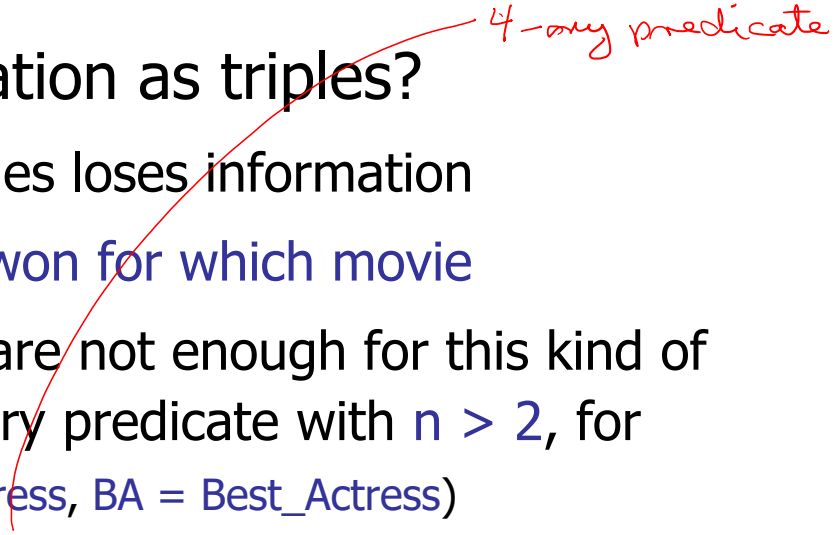
```
<Meryl_Streep> <won_award>  <Best_Supporting_Actress>
<Meryl_Streep> <won_award>  <Best_Actress>
<Meryl_Streep> <won_award_in_year> "1980"
<Meryl_Streep> <won_award_in_year> "1982"
<Meryl_Streep> <won_award_in_year> "2011"
<Meryl_Streep> <won_award_for_film> <Kramer_vs_Kramer>
<Meryl_Streep> <won_award_for_film> <Sophies's_Choice>
<Meryl_Streep> <won_award_for_film> <The_Iron_Lady>
```

Reification 2/8

- How to model complex information as triples?
  - The attempt on the previous slides loses information

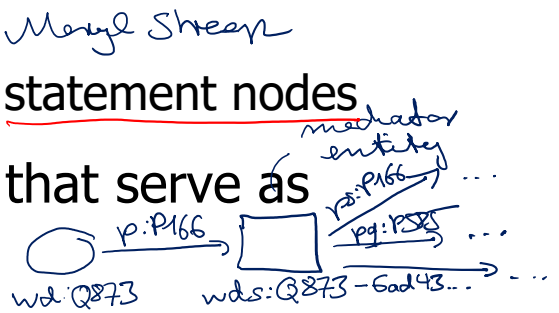
    For example, which award was won for which movie
  - It seems that binary predicates are not enough for this kind of information, but we need an n-ary predicate with n > 2, for example (BSA = Best_Supporting_Actress, BA = Best_Actress)

    <award> <Meryl_Streep> <BSA> "1980" <Kramer_vs_Kramer>
    <award> <Meryl_Streep> <BA> "1983" <Sophie's_Choice>
    <award> <Meryl_Streep> <BA> "2011" <The_Iron_Lady>
  - This would be no problem in a database, but how do we express that in a knowledge graph, as triples?



Reification 3/8

- Mediator entities … in Wikidata, they are called statement nodes
  - We introduce so-called **mediator entities** that serve as a "hub" for the additional information
  - For example, in Wikidata, the information about Meryl Streep's 2011 Oscar (for "The Iron Lady") is represented as follows:




```
@prefix wd: <http://www.wikidata.org/entity/>
@prefix wds: <http://www.wikidata.org/entity/statement/>
@prefix p: <http://www.wikidata.org/prop/>
@prefix ps: <http://www.wikidata.org/prop/statement/>
@prefix pq: <http://www.wikidata.org/prop/qualifier/>

wd:Q873  p:P166  wds:Q873-6ad4311a .  # award statement node
wds:Q873-6ad4311a  ps:P166  wd:Q103618 .  # which award
wds:Q873-6ad4311a  pq:P585  "2011" .  # point in time
wds:Q873-6ad4311a  pq:P1686  wd:Q269810 .  # for work
```

Reification 4/8

- Wikidata's naming scheme
  - Wikidata uses different prefixes for a predicate, depending on whether it stands in relation to a statement node and how

    wdt: normal binary predicate, no statement nodes involved

    p: predicate with statement node as object

    ps: predicate with statement node as subject, leads to primary information of the statement node

    pq: predicate with statement node as subject, leads to additional information of the statement node
  - Note that, technically, the following are all different IRIs

    wdt:P166, p:P166, ps:P166, pq:P166, … [there are more]

    The common suffix tells us that they all "express" the same relation (winning of an award) … it's quite a clever scheme

Reification 5/8

- Reification makes finding the right SPARQL query hard
  - Consider the following simple search request

    Which Oscars did Meryl Streep win and for which movies?

  - The result we are looking for is something like this:

Oscar for Best Supporting Actress	Kramer vs. Kramer
Oscar for Best Actress	Sophie's Choice
Oscar for Best Actress	The Iron Lady

  - On the next slide, you see the correct SPARQL query on the Wikidata knowledge graph

    For Exercise 3 of ES7 you need to find a similar query
    (but you don't need a language FILTER for the ES7 dataset)

Reification 6/8

```
PREFIX rdfs:  <http://www.w3.org/2000/01/rdf-schema#>
PREFIX wdt:   <http://www.wikidata.org/prop/direct/>
PREFIX pq:    <http://www.wikidata.org/prop/qualifier/>
PREFIX ps:    <http://www.wikidata.org/prop/statement/>
PREFIX p:     <http://www.wikidata.org/prop/>
PREFIX wd:    <http://www.wikidata.org/entity/>
SELECT ?movie ?award WHERE {
  wd:Q873     p:P166     ?mediator .
  ?mediator   ps:P166    ?award_id .
  ?mediator   pq:P1686   ?movie_id .
  ?award_id   wdt:P31    wd:Q19020 .
  ?award_id   rdfs:label ?award . FILTER (LANG(?award) = "en")
  ?movie_id   rdfs:label ?movie . FILTER (LANG(?movie) = "en")
}
```

Reification 7/8

- Understand what's hard about finding this query
  - Knowing the right prefix definitions

    PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
  - Knowing the right entity names

    wd:Q873 ("Meryl Streep"), wd:Q19020 ("Academy Award")
  - Knowing the right predicate names (very hard, even for experts)

p:P166	"won award"	from awardee to mediator
ps:P166	"won award"	from mediator to award entity
pq:P1686	"for work"	from mediator to movie
wdt:P31	"instance of"	from "instance" to "class"
  - Knowing the syntax for filtering by language (not needed for ES7)

    FILTER (LANG(?award) = "en")

Reification 8/8

- Autocompletion
  - In our group, we develop QLever, a powerful RDF database

    https://github.com/ad-freiburg/qlever

    We have many thesis and project topics related to QLever → ideal if you want to learn how to contribute to an open source project with many applications and real users
  - The QLever UI offers context-sensitive **autocompletion**, which helps significantly with query formulation

    https://qlever.dev/wikidata

    In Lecture 8, we will learn how to design and implement a simple autocompletion mechanism (and in Lectures 9 and 10, we will implement a web app that combines everything we have done so far)

SPARQL to SQL Translation 1/5

- Motivation
  - We want to translate a given SPARQL query to a SQL query that gives the desired results on a given database
  - In the following example, we use **one table per predicate**

    CREATE TABLE acted_in(person TEXT, film TEXT)
    CREATE TABLE married_to(person1 TEXT, person2 TEXT)

    Note: all elements from one table are from one predicate, so we don't need to store the relation name in the table

    For ES7, use **one big table** for all the data, with three columns named **subject**, **predicate**, **object**

    This is deliberately different from how we do it in the lecture, so that you have to do some thinking yourself

SPARQL to SQL Translation 2/5

- Example
  - SPARQL query

```
SELECT ?person1 ?person2 ?film WHERE {
   ?person1   <acted_in>     ?film  .
   ?person2   <acted_in>     ?film  .
   ?person1   <married_to>   ?person2  .
}
```

  - Corresponding SQL query:

```
SELECT
FROM
WHERE
```

SPARQL to SQL Translation 3/5

- Algorithm
  - It is up to you in ES7, to design a generic algorithm that works for arbitrary basic SPARQL queries or the form

    SELECT <list of variables> WHERE { <list of triples> }

  - The algorithm is not difficult, but requires an understanding of how the data is stored and how SPARQL and SQL work in principle

    In their full generality, SPARQL and SQL are much more complex … for this lecture and for ES7, we focus on the basics

  - On the next slide we provide some key advice for ES 7

    You might want to skip this at first and think about it yourself, and look at it only at the very end

SPARQL to SQL Translation 4/5

- Algorithm, advice for ES7
  - If there are k query triples in the SPARQL query, have k entries in the FROM clause of the SQL query

    FROM wikidata AS w1, wikidata AS w2, ... , wikidata AS wk
  - In your code, for each variable from the SPARQL query, build an **array** of all its occurrences in the query, e.g.

    ?x: w1.subject, w2.object, w5.object
  - Then, when building the SQL query, add the corresponding equalities to the WHERE clause, e.g.

    WHERE w1.subject = w2.object AND w2.object = w5.object

    Note: if ?x occurs m times, m – 1 equalities are enough

SPARQL to SQL Translation 5/5

- More advice for ES7 (Python ⇿ SQL)
  - Executing SQL commands to a SQLite database from within **Python** can be done as follows:

```
import sqlite3
db = sqlite3.connect("example.db")
cursor = db.cursor()
cursor.execute("SELECT * FROM table")
for row in cursor.fetchall():
   print("\t".join(row))
```

Beware: the SQLite commands (starting with a dot) cannot be executed from within Python, you need the CLI for those
[CLI = command-line interface, the "sqlite3" command from the lecture]

Further literature

- Book chapter (gentle introduction to knowledge graphs)
  - H. Bast, J. Kalmbach, T. Klumpp, C. Korzen: Knowledge Graphs

    https://ad-publications.cs.uni-freiburg.de/CHAPTER_knowledge_graphs_BKKK_2023.pdf
- Wikipedia
  - https://en.wikipedia.org/wiki/Knowledge_graph
  - http://en.wikipedia.org/wiki/SPARQL
- Wikidata
  - https://www.wikidata.org
  - https://www.mediawiki.org/wiki/Wikibase/Indexing/RDF_Dump_Format