

Advanced Databases and Information Systems

Lecture 1: Overview, RDF & SPARQL, more SPARQL

October 20, 2026

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

Overview of this lecture

■ Organizational

- Contents of this course Demos + list of topics
- Organization and Style Lectures, exercises, tutorials
- Credits ECTS points + exam info
- Course systems Daphne, Forum, SVN

■ Contents

- Overview Demo and motivation for this course
- RDF and SPARQL Quick intro (recap from DBIS course)
- More SPARQL SPARQL by (many) examples

Exercise Sheet 1: Find queries on several known knowledge graphs and create your own from a given small dataset

Contents of this Course 1/4

- Two demos for starters ... both made in Freiburg

- The **QLever** graph database

Implements the RDF and SPARQL standards (Lectures 1, 2, 3)

RDF = a data model that is a variant of relational databases

SPARQL = the query language for RDF (a variant of SQL)

Let's look at a couple of example datasets and queries

- The **GRASP** question answering system

An agentic AI, driven by a large language model, that interactively explores a graph database like a human would

Let's look at an example question

Contents of this Course 2/4

■ Overview of lectures in this course

Knowledge graphs	intro, indexing, query planning
Efficiency aspects	merge joins, compression
Embeddings	LSI, word embeddings, vector databases
NLP Learning	from logistic regression to language models
Question Answering	from question to query
To be determined	not all topics are fixed yet

As usual for our course, the lectures and exercise sheets will be a good mix of theory and practice

Contents of this Course 3/4

■ Why these topics?

- We believe that the **information system of the future** is a combination of AI and some form of database
- Why not only a database without the AI?

The answer to that one is obvious now: **normal people** want to ask questions in natural language, not queries

For **you**, it is still important to learn a query language, so that you understand how these systems work inside

- Why not only an AI without a database?

In principle, an AI (think: LLM) can also memorize stuff

But it's incredibly inefficient: a brain is not the right system for learning the phone book by heart

Contents of this Course 4/4

■ Prerequisites

- We assume that you have taken the basic course **Databases and Information Systems**, last given by me in WS 25/26

Topics of that course: search engine basics, database basics, intro to knowledge graphs, fuzzy search, web applications, vector space model, simple language models

- We will let you know when a particular lecture from that basic course is important for a lecture from this course

Via an announcement on our Daphne forum (see Slide ???)

For example, part of today's lecture (and the exercise sheet) build on Lecture 7 (knowledge graphs) from the WS 25/26

■ **This is a quick recap** of DBIS, Lecture 7 last given WS 25/26

- In particular, we recap the basic definition of RDF and SPARQL, and what reification is and how it works

But really quickly ... if you have trouble following, make sure to rewatch DBIS Lecture 7 after the lecture and before you work on Exercise Sheet 1

- We then introduce more features of SPARQL

Since this is the first lecture of this course, we will keep it simple and do that by example

The next two lectures will then go a bit deeper

■ The Resource Description Framework (RDF)

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

subject predicate object .

- A small example for an RDF dataset:

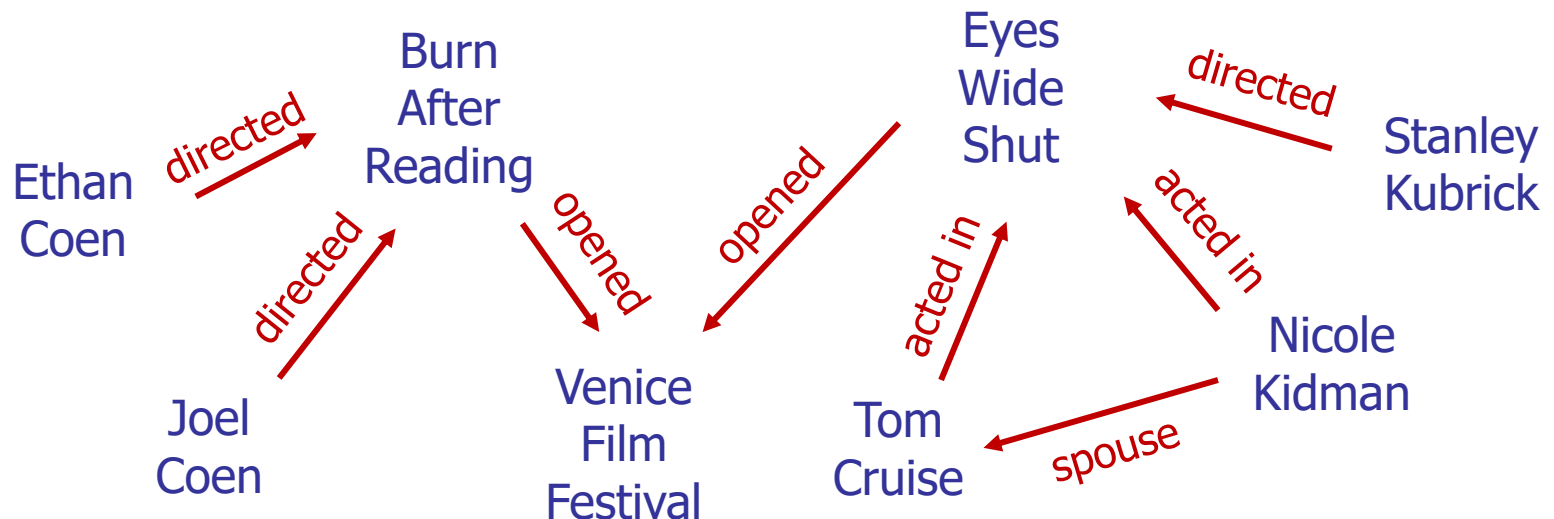
<Nicole_Kidman>	<acted_in>	<Eyes_Wide_Shut>	.
<Tom_Cruise>	<acted_in>	<Eyes_Wide_Shut >	.
<Stanley_Kubrick>	<directed>	<Eyes_Wide_Shut>	.
<Nicole_Kidman>	<spouse>	<Tom_Cruise>	.
<Joel_Coen>	<directed>	<Burn_After_Reading	.
<Ethan_Coen>	<directed>	<Burn_After_Reading>	.
<Eyes_Wide_Shut>	<opened>	<Venice_Film_Festival>	.
<Burn_After_Reading>	<opened>	<Venice_Film_Festival>	.

■ 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



■ 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**):

wd:Q37459 **rdfs:label** "निकोल किडमैन"@hi

wd:Q37459 **wdt:P19** **wd:Q18094**

wd:Q37459 **wdt:P569** "1967-06-20"^^xsd:date

wdt:P19
place of birth

wd:Q18094
Honolulu

wdt:P569
date of birth

Intro to RDF and SPARQL 5/8

■ Three widely used knowledge graphs

all three used in ES1

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

18 **B** triples, 66 **k** predicates



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

217 **B** triples, 115 **k** predicates



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

1.6 **B** triples, 121 predicates



- The standard query language for RDF is SPARQL
 - SPARQL is a variant of SQL, adapted to the (simple) RDF
 - For example, consider the following query on an RDF dataset like the one from our toy example from Slide 9

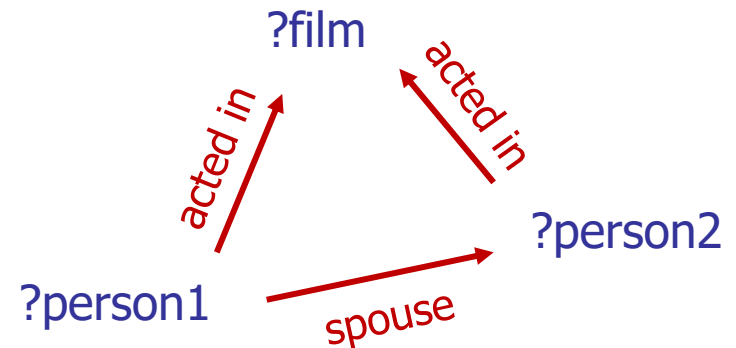
```
SELECT ?person1 ?person2 ?film WHERE {  
    ?person1 <acted_in>    ?film .  
    ?person2 <acted_in>    ?film .  
    ?person1 <spouse>      ?person2 .  
}
```

married couples who acted together in at least one movie

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

■ 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

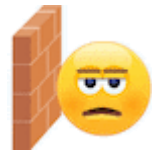
[all power lines in the EU](#)

■ 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

More SPARQL by Example 1/10

■ SPARQL is an open standard

- Defined by the **W3C**, the current official version is SPARQL 1.1

<https://www.w3.org/TR/sparql11-query/>

- The standard defines the complete syntax and semantics

Standards are important, that way there is (almost) no ambiguity regarding that can be written and what it means

- In the following, we will explain some of the most important constructs by example

This is far from complete, but it encompasses everything you need for ES1 ... all of which is  for the exam

More SPARQL by Example 2/10

■ FILTER and language tags

- Example query on Wikidata

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")

}

people born (P19) in Honolulu (Q18094) with their en name

More SPARQL by Example 3/10

■ Property paths with / and *

- Example query on Wikidata (prefix declarations omitted)

```
SELECT DISTINCT ?city ?name ?population WHERE {  
  ?city wdt:P31/wdt:P279* wd:Q515 .  
  ?city wdt:P17 wd:Q183 .  
  ?city wdt:P1082 ?population .  
  ?city rdfs:label ?name FILTER (LANG(?name) = "de")  
}  
ORDER BY DESC(?population)
```

cities (Q515) with country (P17) Germany (Q183), ordered by their population (P1082) in descending (DESC) order

wdt:P31/wdt:P279* stands for one edge "instance of" (P31) followed by zero or more (*) edges "subclass of" (P279)

One result per path, hence the DISTINCT

■ Single OPTIONAL

- Variant of the query on Slide 17

```
SELECT DISTINCT ?city ?name ?population WHERE {  
  ?city wdt:P31/wdt:P279* wd:Q515 .  
  ?city wdt:P17 wd:Q183 .  
  ?city wdt:P1082 ?population .  
  OPTIONAL { ?city rdfs:label ?name FILTER (LANG(?name) = "hi") }  
}  
ORDER BY DESC(?population)
```

German cities with their Hindi name if they have one (if they don't, the city still appears in the result)

■ Multiple OPTIONALs

- Variant of the query on Slide 17 and 18

```
SELECT DISTINCT ?city ?name ?population WHERE {  
  ?city wdt:P31/wdt:P279* wd:Q515 .  
  ?city wdt:P17 wd:Q183 .  
  ?city wdt:P1082 ?population .  
  OPTIONAL { ?city rdfs:label ?name FILTER (LANG(?name) = "hi") }  
  OPTIONAL { ?city rdfs:label ?name FILTER (LANG(?name) = "ar") }  
}  
ORDER BY DESC(?population)
```

German cities with their Hindi name if they have one, and their Arabic name as fallback (and if they have neither, the city still appears in the result)

More SPARQL by Example 6/10

■ ORDER BY and LIMIT

- Variant of the query on Slide 17 and 18 and 19

```
SELECT DISTINCT ?city ?name ?population WHERE {  
  ?city wdt:P31/wdt:P279* wd:Q515 .  
  ?city wdt:P17 wd:Q183 .  
  ?city wdt:P1082 ?population .  
  ?city rdfs:label ?name FILTER (LANG(?name) = "de")  
}  
ORDER BY DESC(?population)  
LIMIT 10
```

The ten most populous German cities

Note that the LIMIT must come after the ORDER BY

More SPARQL by Example 7/10

■ GROUP BY and aggregation

- Example query on dblp

PREFIX **dblp:** <https://dblp.org/rdf/schema#>

SELECT ?venue ?name

 (**COUNT**(?paper) **AS** ?num_papers) **WHERE** {
 ?paper **dblp:**publishedInStream ?venue .
 ?venue **rdfs:**label ?name .
}

GROUP BY ?venue ?name

ORDER BY DESC(?num_papers)

Venues with the number of papers published in it

Note: a variable like ?paper that is not grouped, either has to be omitted in the SELECT or aggregated (COUNT, SUM, AVG, ...)

■ Sub-SELECT

- Variant of the query on Slide 21

```
PREFIX dblp: <https://dblp.org/rdf/schema#>
SELECT ?venue ?name ?num_papers WHERE {
  {
    SELECT ?venue (COUNT(?paper) AS ?num_papers) WHERE {
      ?paper dblp:publishedInStream ?venue .
    } GROUP BY ?venue
  }
  ?venue rdfs:label ?name .
}
```

ORDER BY DESC(?num_papers)

Same result as on Slide 21, but more efficient (we add ?name once per venue, after the aggregation)

More SPARQL by Example 9/10

■ BIND

- Variant of the query on Slides 21 and 22

```
PREFIX dblp: <https://dblp.org/rdf/schema#>
SELECT ?venue ?name ?num_papers WHERE {
  {
    SELECT ?venue (COUNT(?paper) AS ?num_papers) WHERE {
      ?paper dblp:publishedInStream ?venue .
    } GROUP BY ?venue
  }
  BIND (REPLACE(STR(?venue), ".*/", "") AS ?name)
}
ORDER BY DESC(?num_papers)

# BIND adds an additional column, in this case with the part of
the venue IRI after the last /
```

■ Spatial join

- Example query on OpenStreetMap

```
PREFIX osmkey: <https://www.openstreetmap.org/wiki/Key:>
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX geof: <http://www.opengis.net/def/function/geosparql/>
PREFIX unit: <http://qudt.org/vocab/unit/>
SELECT ?building ?building_wkt WHERE {
  ?building osmkey:building ?anything .
  ?building geo:hasGeometry/geo:asWKT ?building_wkt .
  FILTER (geof:distance("POINT(7.842 47.998)"
    ^^geo:wktLiteral,?building_wkt,unit:M) <= 500)
}
# Buildings in a 500m radius around Freiburg Hauptbahnhof
```

Hints for Exercise Sheet 1 1/2

■ Exercise 1: finding the right queries

- Use the three SPARQL endpoints at <https://adbis.qllever.dev/wikidata>, [OpenStreetMap](https://adbis.qllever.dev/OpenStreetMap), [dblp](https://adbis.qllever.dev/dblp)
- Use the SPARQL autocompletion to find out which predicates or objects are possible at a certain query position

Alternatively, write exploratory SPARQL queries

- Construct your queries incrementally

That is, first try to get a (small) part of the query right, and then work yourself up from there (or combine)

Hints for Exercise Sheet 1 2/2

■ Exercise 2: set up your own SPARQL endpoint

- Install QLever

Follow the instructions on <https://docs.qllever.dev/quickstart/>

Should work fine for Ubuntu, Debian, and macOS

Might also work for other Ubuntu-derived distributions

- Build an index and start a server for the given data

You can use the provided "Qleverfile" (QLever's config file)

- If you encounter installation problems ask your favorite AI or on the forum

But please don't ask your favorite AI to do the exercise sheet for you (you won't have AI help during the exam)

* To compensate for the overtime

■ Lectures

- Tuesday 14:**05** – 15:**55** h in SR 101-1-9/13

With one **break** in between at around 15:00 h

- 13 lectures altogether (last one on February 9)

No lecture on Oct 27*, Xxx ??* and Dec 29, Jan 5 (holidays)

- All lectures are recorded and live-streamed

Slides + Audio + Video ... editing: Alexander Monneret

- You find all the course materials on our Wiki

Recordings, slides, code from the lecture, exercise sheets + specifications + design suggestions, master solutions, ...

Also in the SVN, subfolder `/public` (except for the recordings)

Organization and Style 2/6

■ Exercise sheets

- The exercise sheets are an important part of the course
- One sheet per lecture (except the last), altogether 12 sheets

The deadline is always at 12:00 h before the next lecture

- The exercises are **voluntary**

That is, you do **not** need 50% of the points to get your Studienleistung



- You can work in **groups** if you like

But only one of you should commit solutions

- **Plagiarism** is still forbidden

It's meaningless and costs our tutors work, please don't do it

■ Tutorials

- There will **no** time slots for tutorials

We have tried it in 100 variants already, only few people come and those who come are passive → not worth the effort

- There will be **no** weekly Q&A either

We have tried that, too, same problem

- If you submit an exercise sheet and you ask for **feedback**, you will receive it by Friday after the submission deadline

Precondition: you have to write in your **experiences.md** that you want feedback and which kind of feedback you want

- There is a **forum** for all kinds of questions and you can make **individual appointments** with your tutor → [infos in Lecture 2](#)

■ Style of the lectures

- I will provide: motivation, definitions, examples, **live code**

The emphasis is on: basic ideas, intuition, inspiration

Working out the details is **your** job ... you only really learn stuff by doing it yourself

- One topic per lecture + self contained

We provide all the materials you need for the sheets and the exam ... the literature pointers at the end are optional

- We are nice if you are nice

We invest a lot of work in this course; please make sure that you do your part of the work (learning and paying attention)

■ What you should **learn** in this course

- Two kinds of understanding are important:

A. Understand the concepts in depth

B. Understand how to do things in practice

- At the university you often learn deep theory (**A**), but you don't learn where and how to apply it (**B**)
- At other institutions you often learn how to do things (**B**), but you don't really understand why it works and you are not really able to adapt or extend it to new situations (**A**)
- In our courses, we give **A** and **B** equal importance, you will learn about both, and the exam will also test both

■ Master solutions

- After the deadline for each sheet, the master solutions for this sheet will be published

On the Wiki and in the SVN under /solutions

Important: these master solutions are strictly **for your personal use only**, now and in the future

Under no circumstances may you pass them on to others or let others access them from your machine

■ Amount of Work / ECTS points

- This course yields **6 ECTS** points = costs **180** working hours
Lectures (≈ 30 hours) + exercise sheets + exam preparation
- Work per exercise sheet, options
 - A. **= 0 hours** per ES if you don't do them
 - B. **~ 8 hours** per ES if you do them, on average
 - C. **a lot more** per ES if you lack basic prerequisites*

* We always have a fair share of students who have significant gaps in their basic mathematical understanding or in programming

→ if that applies to you, you are still welcome to do this course, and you could learn a lot, but **please be prepared to have to invest much more work and please do not blame us**

■ "Studienleistung"

- All you need to do to get the Studienleistung for this course

Register on Daphne and register for the exam

This is not (yet) urgent ... the deadline is on **17.01.2027**

- Nevertheless, you get points for the exercise sheets

Whoever works on a sheet will probably want to know how well they did (and get valuable feedback)

■ "Prüfungsleistung" (Exam)

- There is a **written** exam in the end

The date will be fixed in the next months

- There will be four tasks à 25 points → 100 points max
- The final grade for your course is linear in the #points:

50 – 54:	4.0	55 – 59:	3.7	60 – 64:	3.3
65 – 69:	3.0	70 – 74:	2.7	75 – 79:	2.3
80 – 84:	2.0	85 – 89:	1.7	90 – 94:	1.3
95 – 100:	1.0				

- The exam will test **all** of the following three:

Basic knowledge, practical application, deeper understanding

Superficial understanding will not be enough to pass

■ **Daphne**, our course management system

- There is a link on the course Wiki, **please register!**
- In Daphne, you have an overview over the following:
 - Your tutor
 - Your points
 - Link to the [Forum](#) see slide 25
 - Link to the [SVN](#) see slide 26
 - Link to our [Build System](#) see slide 27

■ **Forum** for this course

- There is a link on the Wiki and on your Daphne page
- Please ask whenever something is not clear

Please carefully read our guidelines

<https://ad-wiki.informatik.uni-freiburg.de/teaching/Manuals/AskingOnAForum>

If you ignore these guidelines, you risk getting suboptimal answers or no answer at all. We are very eager to help you, but **you have to help us so that we can help you**

- Robin or a tutor or myself will usually answer very quickly

If for some reason you feel that you need help beyond the forum, you can contact your tutor for a **personal meeting**, more about that in Lecture 2

■ **SVN** = Subversion <http://subversion.apache.org>

- A central repository of all course data on our servers
- Typical operations: **svn add**, **svn commit**, **svn update**
- Complete history of all previous versions of the files
- In this course, we will use it for the following:

Your submissions for the exercise sheets

The feedback from your tutors

Slides, code from lecture, exercise sheets, master solutions

This is an advanced course; we expect that you have either seen this already or figure it out yourselves

■ Our continuous build system

- You can (and should) use it to check whether the code that you have uploaded to our SVN passes all checks

In particular, if it compiles

If it is free from checkstyle errors

And if all tests run through without errors

Not relevant for Exercise Sheet 1 yet

References

■ Databases and Information Systems, Basic Course

This is a **prerequisite** for this course

In WS starting in even years: given by Prof. Bödecker

In WS starting in odd years: given by Prof. Bast

See <https://ad-wiki.informatik.uni-freiburg.de/teaching>

■ Articles and websites relevant for this lecture

https://en.wikipedia.org/wiki/Resource_Description_Framework

<https://en.wikipedia.org/wiki/SPARQL>

<https://adbis.qllever.dev>