Informatische Werkzeuge in den Geistes- und Sozialwissenschaften 2

Prof. Dr. Michael Kohlhase

Professur für Wissensrepräsentation und -verarbeitung
Informatik, FAU Erlangen-Nürnberg
Michael.Kohlhase@FAU.de

2024-02-08

Chapter 8
Semester Change-Over

8.1 Administrativa

Prerequisites

- **Formal Prerequisite:** IWGS-1 (If you did not take it, read the notes)
- **General Prerequisites:** Motivation, interest, curiosity, hard work.
  nothing else! (apart from IWGS-1)
  We will teach you all you need to know
- You can do this course if you want! (we will help)

Assessment, Grades

- **Grading Background/Theory:** Only modules are graded! (by the law)
  - Module "DH-Einführung" (DHE) $\widehat{=}$ courses IWGS1/2, DH-Einführung.
  - DHE module grade $\leadsto$ pass/fail determined by "portfolio" $\widehat{=}$ collection of contributions/assessments.
- **Assessment Practice:** The IWGS assessments in the "portfolio" consist of
  - weekly homework assignments, (practice IWGS concepts and tools)
  - 60 minutes exam directly after lectures end: July 27. 2024.
- **Retake Exam:** 60 min exam at the end of the exam break.(October. 12. 2024)

IWGS Homework Assignments

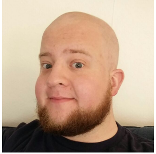
- **Homeworks:** will be small individual problem/programming/system assignments
  - but take time to solve (at least read them directly $\leadsto$ questions)
  - group submission if and only if explicitly permitted.
- ⚠ Without trying the homework assignments you are unlikely to pass the exam.
- **Admin:** To keep things running smoothly
  - Homeworks will be posted on StudOn.
  - Sign up for IWGS under `https://www.studon.fau.de/frm5075965.html`.
  - Homeworks are handed in electronically there. (plain text, program files, PDF)
  - Go to the tutorials, discuss with your TA! (they are there for you!)
- **Homework Discipline:**
  - Start early! (many assignments need more than one evening's work)
  - Don't start by sitting at a blank screen (talking & study group help)
  - Humans will be trying to understand the text/code/math when grading it.

IWGS Tutorials

- Weekly tutorials and homework assignments (first one in week two)

  Tutor: (Doctoral Student in CS)
  - Jonas Betzendahl: `jonas.betzendahl@fau.de`

  They know what they are doing and really want to help you learn! (dedicated to DH)



- **Goal 1:** Reinforce what was taught in class (important pillar of the IWGS concept)
- **Goal 2:** Let you experiment with Python (think of them as Programming Labs)
- **Life-saving Advice:** go to your tutorial, and prepare it by having looked at the slides and the homework assignments
- **Inverted Classroom:** the latest craze in didactics (works well if done right)
  in IWGS: Lecture + Homework assignments + Tutorials $\widehat{=}$ inverted classroom

Textbook, Handouts and Information, Forums, Videos

- **No Textbook:** but lots of online python tutorials on the web.
- Course notes will be posted at `http://kwarc.info/teaching/IWGS` (see references)
  - I mostly prepare/adapt/correct them as we go along.
  - please e-mail me any errors/shortcomings you notice. (improve for the group)
- The lecture videos of WS 2020/21 are at https://www.fau.tv/course/id/2350 (not much changed)
- Matrix chat at `#iwgs:fau.de` (via IDM) (instructions)
- **StudOn Forum:** `https://www.studon.fau.de/frm5075965.html` for
  - announcements, homeworks (my view on the forum)
  - questions, discussion among your fellow students (your forum too, use it!)
- If you become an active discussion group, the forum turns into a valuable resource!

Experiment: Learning Support with KWARC Technologies

- **My research area:** Deep representation formats for (mathematical) knowledge
- **One Application:** Learning support systems (represent knowledge to transport it)
- **Experiment:** Start with this course (Drink my own medicine)
  1. Re-represent the slide materials in OMDoc (Open Mathematical Documents)
  2. Feed it into the ALeA system (`http://courses.voll-ki.fau.de`)
  3. Try it on you all (to get feedback from you)
- Research tasks
  - help me complete the material on the slides (what is missing/would help?)
  - I need to remember "what I say", examples on the board. (take notes)
- Benefits for you (so why should you help?)
  - you will be mentioned in the acknowledgements (for all that is worth)
  - you will help build better course materials (think of next-year's students)

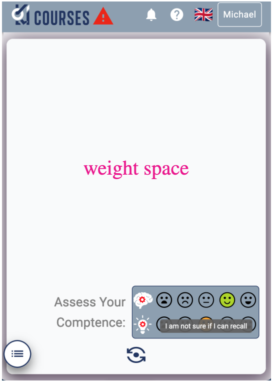
VoLL-KI Portal at `https://courses.voll-ki.fau.de`

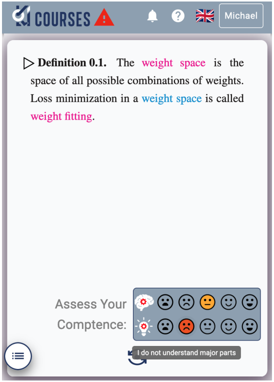
- **Portal for ALeA Courses:** `https://courses.voll-ki.fau.de`

- **AI-1 in ALeA:** `https://courses.voll-ki.fau.de/course-home/ai-1`
  - All details for the course.
  - recorded syllabus (keep track of material covered in course)
  - syllabus of the last semester (for over/preview)
- **ALeA Status:** The ALeA system is deployed at FAU for over 1000 students taking six courses
  - (some) students use the system actively (our logs tell us)
  - reviews are mostly positive/enthusiastic (error reports pour in)

New Feature: Drilling with Flashcards

- Flashcards challenge you with a task (term/problem) on the front...





...and the definition/answer is on the back.

- Self-assessment updates the learner model (before/after)
- **Idea:** Challenge yourself to a card stack, keep drilling/assessing flashcards until the learner model eliminates all.
- **Bonus:** Flashcards can be generated from existing semantic markup (educational equivalent to free beer)

Practical recommendations on Lecture Videos

- **Excellent Guide:** [Nor+18a] (german Version at [Nor+18b])

- Normally intended for "offline students" $\widehat{=}$ everyone during Corona times.

Software/Hardware tools

- You will need computer access for this course
- we recommend the use of standard software tools
  - find a text editor you are comfortable with (get good with it) A text editor is a program you can use to write text files. (not `MSWord`)
  - any operating system you like (I can only help with `UNIX`)
  - Any browser you like (I use `FireFox`: less spying)
- **Advice:** learn how to touch-type NOW (reap the benefits earlier, not later)
  - you will be typing multiple hours/week in the next decades
  - touch-typing is about twice as fast as "system eagle".
  - you can learn it in two weeks (good programs)

Outline of IWGS-II:

- Databases
  - CRUD operations, querying, and python embedding
  - XML and JSON for file based data storage

Outline of IWGS-II:

- Databases
  - CRUD operations, querying, and python embedding
  - XML and JSON for file based data storage
- BooksApp: a Books Application with persistent storage

Outline of IWGS-II:

- Databases
  - CRUD operations, querying, and python embedding
  - XML and JSON for file based data storage
- BooksApp: a Books Application with persistent storage
- Image processing
  - Basics
  - Image transformations, Image Understanding

Outline of IWGS-II:

- Databases
  - CRUD operations, querying, and python embedding
  - XML and JSON for file based data storage
- BooksApp: a Books Application with persistent storage
- Image processing
  - Basics
  - Image transformations, Image Understanding
- Ontologies, semantic web, and WissKI
  - Ontologies (inference $\leadsto$ get out more than you put in)
  - semantic web Technologies (standardize ontology formats and inference)
  - Using semantic web Tech for cultural heritage research data $\leadsto$ the WissKI System

Outline of IWGS-II:

- Databases
  - CRUD operations, querying, and python embedding
  - XML and JSON for file based data storage
- BooksApp: a Books Application with persistent storage
- Image processing
  - Basics
  - Image transformations, Image Understanding
- Ontologies, semantic web, and WissKI
  - Ontologies (inference $\leadsto$ get out more than you put in)
  - semantic web Technologies (standardize ontology formats and inference)
  - Using semantic web Tech for cultural heritage research data $\leadsto$ the WissKI System
- Legal Foundations of Information Systems
  - Copyright & Licensing
  - Data Protection (GDPR)

IWGS-II Project

- **Idea:** Consolidate the techniques from IWGS-I and IWGS-II into a prototypical information system for Art History @ FAU. (Practical Digital Humanities)
- **A Running Example:** Research image + metadata collection "Bauernkirmes" provided by Prof. Peter Bell

IWGS-II Project

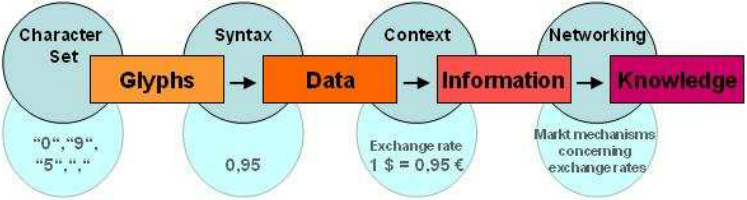
- **Idea:** Consolidate the techniques from IWGS-I and IWGS-II into a prototypical information system for Art History @ FAU. (Practical Digital Humanities)
- **A Running Example:** Research image + metadata collection “Bauernkirmes” provided by Prof. Peter Bell
- **What will you do?:** Build a web-based image/data manager, test image algorithms, annotate ontologically, ...
- **How will we organize this:** Mostly via the group homework assignments (together they will make the project)

Chapter 9
Databases

9.1 Introduction

Databases, Data, Information, and Knowledge

- **Definition 1.1.** Discrete, objective facts or observations, which are unorganized and uninterpreted are called data (singular datum).
- According to Probst/Raub/Romhardt [PRR97]



- **Example 1.2.** The height of Mt. Everest (8.848 meters) is a datum.

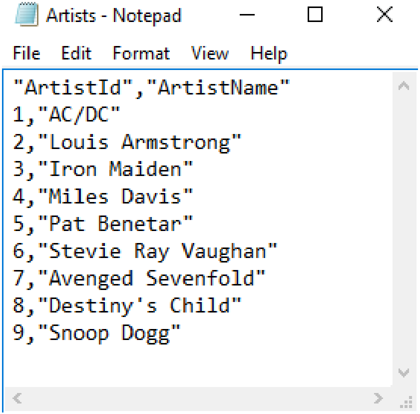
  Definition 1.3. A database is an organized collection of data, stored and accessed electronically from a computer system.

Storing Data Electronically

- Four conventional ways of storing data: (mileage varies)
  - In the computer's memory (RAM) (very fast (+), random access (+), but not persistent (-))

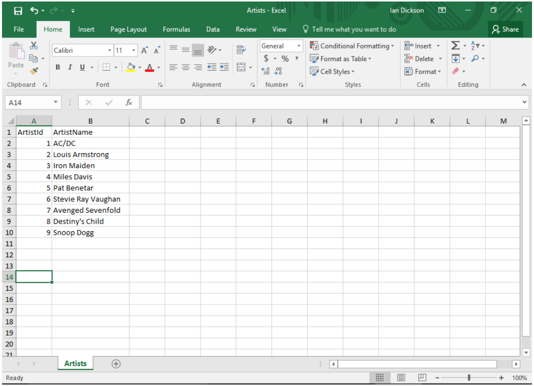
Storing Data Electronically

- Four conventional ways of storing data: (mileage varies)
  - In the computer's memory (RAM) (very fast (+), random access (+), but not persistent (-))
  - In a text file (persistent (+), fast (+), sequential access ( ), unstructured ( ))



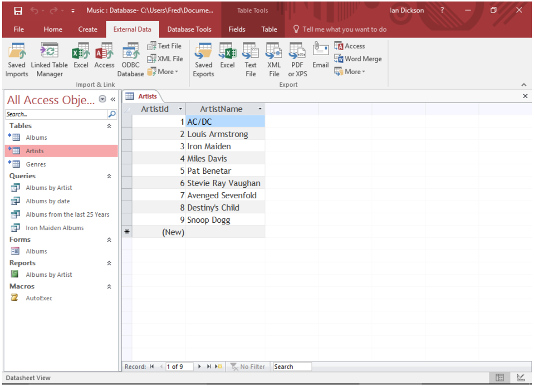
Storing Data Electronically

- Four conventional ways of storing data: (mileage varies)
  - In the computer's memory (RAM) (very fast (+), random access (+), but not persistent (-))
  - In a text file (persistent (+), fast (+), sequential access ( ), unstructured ( ))
  - In a spreadsheet (persistent (+), 2D-structured (+-), relations (+), slow (-))



Storing Data Electronically

- Four conventional ways of storing data: (mileage varies)
  - In the computer's memory (RAM) (very fast (+), random access (+), but not persistent (-))
  - In a text file (persistent (+), fast (+), sequential access ( ), unstructured ( ))
  - In a spreadsheet (persistent (+), 2D-structured (+-), relations (+), slow (-))
  - In a database (persistent (+), scalable (+), relations(+), managed (+), slow (-))



Storing Data Electronically

- Four conventional ways of storing data: (mileage varies)
  - In the computer's memory (RAM) (very fast (+), random access (+), but not persistent (-))
  - In a text file (persistent (+), fast (+), sequential access ( ), unstructured ( ))
  - In a spreadsheet (persistent (+), 2D-structured (+-), relations (+), slow (-))
  - In a database (persistent (+), scalable (+), relations(+), managed (+), slow (-))
- Databases constitute the most scalable, persistent solution.

9.2 Relational Databases

(Relational) Database Management Systems

- **Definition 2.1.** A database management system (DBMS) is program that interacts with end users, applications, and a database to capture and analyze the data and provides facilities to administer the database.
- There are different types of DBMS, we will concentrate on relational ones.
- **Definition 2.2.** In a relational database management system (RDBMS), data are represented as tables: every datum is represented by a row (also called database record), which has a value for all columns (also called an column attribute) or field). A null value is a special "value" used to denote a missing value.
- **Remark:** Mathematically, each row is an *n* tuple of values, and thus a table an *n*-ary relation. (useful for standardizing RDBMS operations)
- **Example 2.3 (Bibliographic Data).**

LastN	FirstN	YOB	YOD	Title	YOP	Publisher	City
Twain	Mark	1835	1910	Huckleberry Finn	1986	Penguin USA	NY
Twain	Mark	1835	1910	Tom Sawyer	1987	Viking	NY
Cather	Willa	1873	1947	My Antonia	1995	Library of America	NY
Hemingway	Ernest	1899	1961	The Sun Also Rises	1995	Scribner	NY
Wolfe	Thomas	1900	1938	Look Homeward, Angel	1995	Scribner	NY
Faulkner	William	1897	1962	The Sound and the Furry	1990	Random House	NY

- **Definition 2.4.** Tables are identified by table name and individual components of records by column name.

Open-Source Relational Database Management Systems

- **Definition 2.5.** MySQL is an open source RDBMS. For simple data sets and web applications MySQL is a fast and stable multi user system featuring an SQL database server that can be accessed by multiple clients.
- **Definition 2.6.** PostgreSQL is an open source RDBMS with an emphasis on extensibility, standards compliance, and scalability.
- **Definition 2.7.** SQLite is an embeddable RDBMS. Instead of a database server, SQLite uses a single database file, therefore no server configuration is necessary.
- **Remark:** At the level we use SQL in IWGS, all are equivalent.
- We will use SQLite in IWGS, since it is easiest to install and configure.

Working with SQLite (via the SQLite shell)

- In IWGS we will use SQLite, since it is very lightweight, easy to install, but feature complete, and widely used.
- Download SQLite at `https://www.sqlite.org/download.html`,
  - e.g. sqlite–dll–win64–x64–3280000.zip for windows.

Working with SQLite (via the SQLite shell)

- In IWGS we will use SQLite, since it is very lightweight, easy to install, but feature complete, and widely used.
- Download SQLite at `https://www.sqlite.org/download.html`,
  - e.g. sqlite–dll–win64–x64–3280000.zip for windows.
  - unzip it into a suitable location, start sqlite3.exe there
  - this opens a command line interpreter: the SQLite shell. (all DBs have one) test it with .help that tells you about more "dot commands".

```
> sqlite3
SQLite version 3.24.0 2018−06−04 19:24:41
Enter ".help" for usage hints.
Connected to a transient in−memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .help
.archive ... Manage SQL archives: ".archive −−help" for details
.auth ON|OFF Show authorizer callbacks
[...]
```

Working with SQLite (via the SQLite shell)

- In IWGS we will use SQLite, since it is very lightweight, easy to install, but feature complete, and widely used.
- Download SQLite at `https://www.sqlite.org/download.html`,
  - e.g. sqlite–dll–win64–x64–3280000.zip for windows.
  - unzip it into a suitable location, start sqlite3.exe there
  - this opens a command line interpreter: the SQLite shell. (all DBs have one) test it with .help that tells you about more "dot commands".
  - If you have a database file books.db from 3.8, use that.

```
> sqlite3 books.db
SQLite version 3.24.0 2018-06-04 19:24:41
Enter ".help" for usage hints.
> .tables
Books
>select * from Books;
Twain|Mark|1835|1910|Huckleberry Finn|1986|Penguin USA|NY
Twain|Mark|1835|1910|Tom Sawyer|1987|Viking|NY
Cather|Willa|1873|1947|My Antonia|1995|Library of America|NY
Hemingway|Ernest|1899|1961|The Sun Also Rises|1995|Scribner|NY
Wolfe|Thomas|1900|1938|Look Homeward, Angel|1995|Scribner|NY
Faulkner|William|1897|1962|The Sound and the Furry|1990|Random House |NY
Tolkien|John Ronald Reuel|1892|1973|The Hobbit|1937|George Allen  Unwin|UK
```

Working with SQLite (via the SQLite shell)

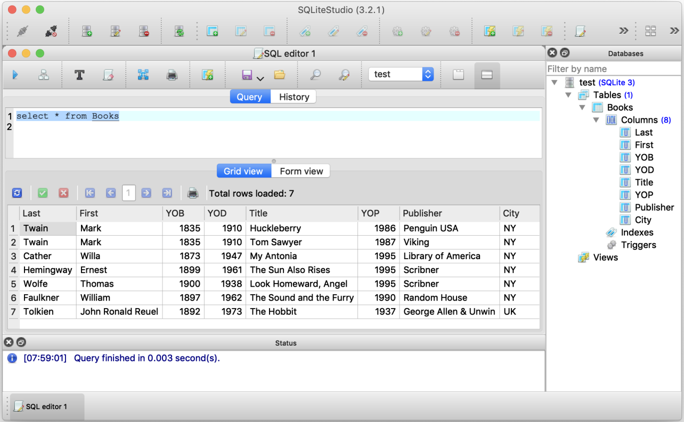
- In IWGS we will use SQLite, since it is very lightweight, easy to install, but feature complete, and widely used.
- Download SQLite at `https://www.sqlite.org/download.html`,
  - e.g. sqlite–dll–win64–x64–3280000.zip for windows.
  - unzip it into a suitable location, start sqlite3.exe there
  - this opens a command line interpreter: the SQLite shell. (all DBs have one)
    test it with .help that tells you about more "dot commands".
  - If you have a database file books.db from 3.8, use that.
  - .tables shows the available tables
    select * from Books is SQL (see below); it shows all entries of the Books table.

A Graphical User Interface for SQLite

- **Definition 2.8.** A database browser is a graphical user interface for a RDBMS that (typically) bundles an SQL instruction editor with displays for query results and the database schema in separate windows.

A Graphical User Interface for SQLite

- **Definition 2.9.** A database browser is a graphical user interface for a RDBMS that (typically) bundles an SQL instruction editor with displays for query results and the database schema in separate windows.
- I will sometimes use one for SQLite in the slides: SQLite Studio (lots of others)
  - download from `https://sqlitestudio.pl`



A Graphical User Interface for SQLite

- **Definition 2.10.** A database browser is a graphical user interface for a RDBMS that (typically) bundles an SQL instruction editor with displays for query results and the database schema in separate windows.
- I will sometimes use one for SQLite in the slides: SQLite Studio (lots of others)
  - download from `https://sqlitestudio.pl`
- Everything we can do with this, we can do with the database shell as well. (just looks nicer)

9.3 SQL – A Standardized Interface to RDBMS

SQL: The Structured Query Language

- **Idea:** We need a language for describing all operations of a RDBMSs.
  - **basics**: creating, reading, updating, deleting database components (CRUD)
  - querying: selecting from and inserting into the database
  - **access control**: who can do what in a database
  - **transactions**: ensuring a consistent database state.

  Definition 3.1. SQL, the structured query language is a domain-specific language for managing data held in a RDBMS. SQL instructions are directly executed by the RDBMS to change the database state or compute answers to SQL queries.
-

DDL: Data Definition Language

- **Definition 3.2.** The data definition language (DDL) is a subset of SQL instructions that address the creation and deletion of database objects.
- **Definition 3.3.** The SQL statement **CREATE TABLE**⟪name⟫ (⟪coldefs⟫) creates a table with name ⟪name⟫. ⟪coldefs⟫ are column specifications that specify the columns: it is a comma-separated list of column names and SQL data type. The totality of all column specifications of all tables in a database is called the database schema.
- **Example 3.4 (Creating a Table).** The following SQL statement creates the table from 2.3

```
CREATE TABLE Books (
    LastN varchar(128), FirstN varchar(128),
    YOB int, YOD int, Title varchar(255), YOP int,
    Publisher varchar(128), City varchar(128)
  );
```

- Other **CREATE** statements exist, e.g. **CREATE** DATABASE ⟪name⟫.
- **Definition 3.5.** The SQL statement **DROP** ⟪obj⟫ ⟪name⟫ deletes the database object of class ⟪obj⟫ with name ⟪name⟫.

SQL Data Types (for Column Specifications)

- **Definition 3.6.** SQL specifies data type for values including:
  - VARCHAR (⟪length⟫): character strings, including Unicode, of a variable length is up to the maximum length of ⟪length⟫.
  - BOOL truth values: **true**, **false** and case variants.
  - INT: Integers
  - FLOAT: floating point numbers
  - DATE: dates, e.g. DATE '1999—01—01' or DATE '2000—2—2'
  - TIME: time points in ISO format, e.g. TIME '00:00:00' or time '23:59:59.99'
  - TIMESTAMP: a combination of DATE and TIME (separated by a blank).
  - CLOB (⟪length⟫) (character large object) up to (typically) 2GiB
  - BLOB (⟪length⟫) (binary large object) up to (typically) 2GiB

SQL: Adding Records to Tables

- **Definition 3.7.** SQL provides the **INSERT INTO** command for inserting records into a table. This comes in two forms:
  1. **INSERT INTO** $\langle\langle$table$\rangle\rangle$ **VALUES** ($\langle\langle$vals$\rangle\rangle$); where $\langle\langle$vals$\rangle\rangle$ is a comma-separated list of values given in the order the columns were declared in the **CREATE TABLE** instruction.
  2. **INSERT INTO** $\langle\langle$table$\rangle\rangle$ ($\langle\langle$cols$\rangle\rangle$) **VALUES** ($\langle\langle$vals$\rangle\rangle$) where $\langle\langle$vals$\rangle\rangle$ is a comma-separated list of values given in the order of $\langle\langle$cols$\rangle\rangle$ (a subset of columns) all other fields are filled with **NULL**

SQL: Adding Records to Tables

- **Definition 3.10.** SQL provides the **INSERT INTO** command for inserting records into a table. This comes in two forms:
  1. **INSERT INTO** ⟪table⟫ **VALUES** (⟪vals⟫); where ⟪vals⟫ is a comma-separated list of values given in the order the columns were declared in the **CREATE TABLE** instruction.
  2. **INSERT INTO** ⟪table⟫ (⟪cols⟫) **VALUES** (⟪vals⟫) where ⟪vals⟫ is a comma-separated list of values given in the order of ⟪cols⟫ (a subset of columns) all other fields are filled with **NULL**
- **Example 3.11 (Inserting into the Books Table).** The given the table Books from 3.4 we can add a record with

```
INSERT INTO Books
VALUES ('Tolkien', 'John␣Ronald␣Reuel', 1892, 1973, 'The␣Hobbit', 1937,
        'George␣Allen␣ Unwin', 'UK');
```

SQL: Adding Records to Tables

- **Definition 3.13.** SQL provides the **INSERT INTO** command for inserting records into a table. This comes in two forms:
  1. **INSERT INTO** ⟪table⟫ **VALUES** (⟪vals⟫); where ⟪vals⟫ is a comma-separated list of values given in the order the columns were declared in the **CREATE TABLE** instruction.
  2. **INSERT INTO** ⟪table⟫ (⟪cols⟫) **VALUES** (⟪vals⟫) where ⟪vals⟫ is a comma-separated list of values given in the order of ⟪cols⟫ (a subset of columns) all other fields are filled with **NULL**
- **Example 3.14 (Inserting into the Books Table).** The given the table Books from 3.4 we can add a record with

```
INSERT INTO Books
VALUES ('Tolkien', 'John␣Ronald␣Reuel', 1892, 1973, 'The␣Hobbit', 1937,
        'George␣Allen␣ Unwin', 'UK');
```

- **Example 3.15 (Inserting Partial Data).** Using the second form of the **INSERT** instruction, we can insert partial data. (all we have)

```
INSERT INTO Books (FirstN, LastN, YOB, title, YOP)
VALUES ('Michael', 'Kohlhase', '1964', 'IWGS␣Course␣Notes', '2018');
```

SQL: Deleting Records from Tables

- **Definition 3.16.** The SQL delete statement allows to change existing records.

  DELETE FROM ⟪table⟫ **WHERE** ⟪condition⟫;

- **Example 3.17.** Deleting the record for "Huckleberry Finn".

  DELETE FROM Works **WHERE** Title = 'Huckleberry␣Finn'

- ⚠ If we leave out the **WHERE** clause, all rows are deleted.
- **Note:** There is much more to the **WHERE** clause, we will get to that when we come to SQL querying. (see)

SQL: Updating Records in Tables

- **Definition 3.18.** The SQL update statement allows to change existing records.

  UPDATE ⟪table⟫
  SET ⟪column⟫$_1$ = ⟪value⟫$_1$, ⟪column⟫$_2$ = ⟪value⟫$_2$, . . .
  WHERE ⟪condition⟫;

- **Example 3.19.** Updating the publisher in "Huckleberry Finn".

  UPDATE Books
  SET Publisher = 'Chatto/Windus', YOP = 1884, City = 'London'
  WHERE Title = 'Huckleberry␣Finn'

- ⚠ If we leave out the **WHERE** clause, all rows are updated.

9.4 ER-Diagrams and Complex Database Schemata

Avoiding Redundancy in Databases

- Recall the books table from 2.3:

LastN	FirstN	YOB	YOD	Title	YOP	Publisher	City
Twain	Mark	1835	1910	Huckleberry Finn	1986	Penguin USA	NY
Twain	Mark	1835	1910	Tom Sawyer	1987	Viking	NY
Cather	Willa	1873	1947	My Antonia	1995	Library of America	NY
Hemingway	Ernest	1899	1961	The Sun Also Rises	1995	Scribner	NY
Wolfe	Thomas	1900	1938	Look Homeward, Angel	1995	Scribner	NY
Faulkner	William	1897	1962	The Sound and the Furry	1990	Random House	NY

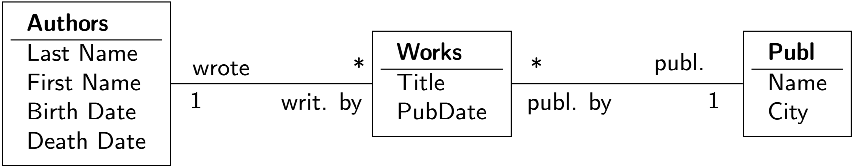
- **Observation:** Some of the fields appear multiple times, e.g. "Mark Twain".
- When the database grows this can lead to scalability problems:
  - in querying: e.g. if we look for all works by Mark Twain
  - in maintenance: e.g. if we want to replace the pen name "Mark Twain" by the real name "Samuel Langhorne Clemens".
- **Idea:** Separate concerns (here Authors, Works, and Publishers) into separate entities, mark their relations.
  - Develop a graphical notation for planning
  - Implement that into the database

Entity Relationship Diagrams

- **Definition 4.1.** An entity relationship diagram (ERD) illustrates the logical structure of a database. It consists of entities that characterize (sets of) objects by their attributes and relations between them.
- **Example 4.2 (An ERD for Books).** Recall the Books table from 2.3:

LastN	FirstN	YOB	YOD	Title	YOP	Publisher	City
Twain	Mark	1835	1910	Huckleberry Finn	1986	Penguin USA	NY
Twain	Mark	1835	1910	Tom Sawyer	1987	Viking	NY
Cather	Willa	1873	1947	My Antonia	1995	Library of America	NY
Hemingway	Ernest	1899	1961	The Sun Also Rises	1995	Scribner	NY
Wolfe	Thomas	1900	1938	Look Homeward, Angel	1995	Scribner	NY
Faulkner	William	1897	1962	The Sound and the Furry	1990	Random House	NY

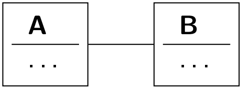
  - **Problem**: We have duplicate information in the authors and publishers
  - **Idea**: Spread the Books information over multiple tables.



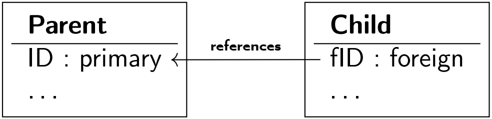
Linking Tables via Primary and Foreign Keys

- **Definition 4.3.** A column in a table can be designated as a primary key, if its values are non-null and unique i.e. all distinct.
- In DDL, we just add the keyword **PRIMARY KEY** to the column specification.
- **Definition 4.4.** A foreign key is a column (or collection of columns) in one table (called the child table) that refers to the primary key in another table (called the reference table or parent table).
- **Intuition:** Together primary keys and foreign keys can be used to link tables or (dually) to spread information over multiple tables.

ERD



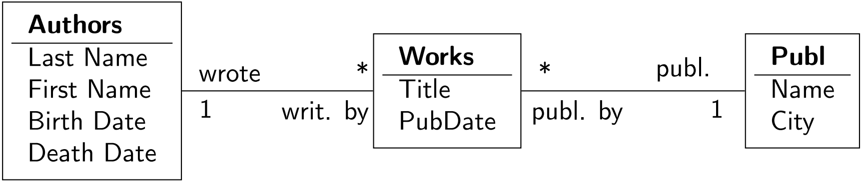
Implementation



- **BTW:** Primary keys are great for identification in the **WHERE** clauses of SQL instructions.

Linking Tables via Primary and Foreign Keys (Example)

- **Example 4.5.** Continuing 4.2, we now implement



by introducing primary keys in the Authors and Publishers tables and referencing them by foreign keys in the Works table.

```
CREATE TABLE Authors (AuthorID int PRIMARY KEY,
    LastN varchar(128), FirstN varchar(128), YOB int, YOD int);

CREATE TABLE Publishers (PublisherID int PRIMARY KEY,
    Name varchar(128), City varchar(128));

CREATE TABLE Works (
    Title varchar(255), YOP int, AuthorID int, PublisherID int,
    FOREIGN KEY(AuthorID) REFERENCES Authors(AuthorID),
    FOREIGN KEY(PublisherID) REFERENCES Publishers(PublisherID));
```

Linking Tables via Primary and Foreign Keys (continued)

- **Example 4.6 (Inserting into the Works Table).** The given the tables Works Authors, and Publishers from 4.5 we can add a record with

```
INSERT INTO Authors VALUES (1, 'Twain', 'Mark', 1835, 1910);
INSERT INTO Publishers VALUES (1, 'Penguin USA', 'NY');
INSERT INTO Works VALUES ('Huckleberry Finn', 1986, 1, 1);

INSERT INTO Publishers VALUES (2,'Viking', 'NY');
INSERT INTO Works VALUES ('Tom Sawyer', 1987, 1, 2);
```

9.5 RDBMS in Python

Using SQLite from Python

- We will use the PySQLite package
  - install it locally with pip install pysqlite for Python 3.
  - use **import** sqlite3 to import the library in your programs.
- Typical Python program with sqlite3:

```
import sqlite3
# Open database connection
db = sqlite3.connect(⟪host⟫,⟪user⟫,⟪pass⟫,⟪DBname⟫)
# prepare a cursor object using cursor() method
cursor = db.cursor()
# execute SQL commands using the execute() method.
cursor.execute("⟪SQL⟫")
⟪dataprocessingcode⟫
# make sure data reaches disk
db.commit()
# disconnect from server
db.close()
```

We will assume this as a wrapper for all code examples below.

Creating Tables in Python

▶ **Example 5.1.** Creating the table of 3.4

```
import sqlite3
# our database file
database = "C:\\sqlite\db\books.db"
# a string with the SQL instruction to create a table
create = """CREATE TABLE Books (
           LastN varchar(128), FirstN varchar(128), YOB int, YOD int,
           Title varchar(255), YOP int,Publisher varchar(128), City varchar(128));"""
insert1 = """INSERT INTO Books
             VALUES ('Twain', 'Mark', '1835', '1910', 'Huckleberry Finn', '1986',
                    'Penguin USA', 'NY');"""
insert2 = """INSERT INTO Books
             VALUES ('Twain', 'Mark', '1835', '1910', 'Tom Sawyer', '1987',
                    'Viking', 'NY');"""
# connect to the SQLIte DB and make a cursor
db = sqlite3.connect(database)
cursor = db.cursor()
# create Books table by executing the cursor
cursor.execute("DROP␣TABLE␣Books;")
cursor.execute(create)
cursor.execute(insert1)
cursor.execute(insert2)
db.commit() # commit to disk
db.close() # clean up by closing
```

To commit or not to commit?

- **Recall:** SQLite computes with tables in memory but uses files for persistence.
- **Also Recall:** Memory access is 100-10.000 times as fast as file access.
- **Idea 1:** Keep tables in memory, write to file only when necessary.
- **Idea 2:** Give the user/programmer control over when to write to file
  - db = sqlite3.connect(⟪file⟫) connects to ⟪file⟫, but computes in memory,
  - db.commit() writes in-memory changes to ⟪file⟫.
- **Problem:** We can have multiple database connections to the same database file in parallel, there may be race conditions and conflicts.
- **Our Solution:** Commit often enough! (your responsibility/fault)
- **General Solution:** RDBMS offer database transactions. (not covered in IWGS)
- **Lazy Solution:** Set the connection to autocommit mode: (system decides) sqlite3.connect(⟪file⟫,isolation_level = None)

9.6 Excursion: Programming with Exceptions in Python

How to deal with Errors in Python

- **Theorem 6.1 (Kohlhase's Law).**

How to deal with Errors in Python

- **Theorem 6.5 (Kohlhase's Law).** *I can be an idiot, and I do make mistakes!*
- **Corollary 6.6.** *Programming languages need a good way to deal with all kinds of errors!*

How to deal with Errors in Python

- **Theorem 6.9 (Kohlhase's Law).** *I can be an idiot, and I do make mistakes!*
- **Corollary 6.10.** *Programming languages need a good way to deal with all kinds of errors!*
- **Definition 6.11.** An exception is a special Python object. Raising an exception e terminates computation and passes e to the next higher level.

How to deal with Errors in Python

- **Theorem 6.13 (Kohlhase's Law).** *I can be an idiot, and I do make mistakes!*
- **Corollary 6.14.** *Programming languages need a good way to deal with all kinds of errors!*
- **Definition 6.15.** An exception is a special Python object. Raising an exception e terminates computation and passes e to the next higher level.
- **Example 6.16 (Division by Zero).** The Python interpreter reports unhandled exceptions.

```
>>> -3 / 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
Zerodivisionerror: division by zero
```

How to deal with Errors in Python

- **Theorem 6.17 (Kohlhase's Law).** *I can be an idiot, and I do make mistakes!*
- **Corollary 6.18.** *Programming languages need a good way to deal with all kinds of errors!*
- **Definition 6.19.** An exception is a special Python object. Raising an exception e terminates computation and passes e to the next higher level.
- **Example 6.20 (Division by Zero).** The Python interpreter reports unhandled exceptions.
- Exceptions are first class citizens in Python, in particular they
  - are classified by their classes in a hierarchy.
  - exception classes can be defined by the user (they inherit from the Exception class)

```
class DivByZero (Exception)
    pass
```

How to deal with Errors in Python

- **Theorem 6.21 (Kohlhase's Law).** *I can be an idiot, and I do make mistakes!*
- **Corollary 6.22.** *Programming languages need a good way to deal with all kinds of errors!*
- **Definition 6.23.** An exception is a special Python object. Raising an exception e terminates computation and passes e to the next higher level.
- **Example 6.24 (Division by Zero).** The Python interpreter reports unhandled exceptions.
- Exceptions are first class citizens in Python, in particular they
  - are classified by their classes in a hierarchy.
  - exception classes can be defined by the user (they inherit from the Exception class)
  - can be raised when an abnormal condition appears

```
if denominator == 0 :
    raise DivByZero
else
    ⟨⟨computation⟩⟩
```

How to deal with Errors in Python

- **Theorem 6.25 (Kohlhase's Law).** *I can be an idiot, and I do make mistakes!*
- **Corollary 6.26.** *Programming languages need a good way to deal with all kinds of errors!*
- **Definition 6.27.** An exception is a special Python object. Raising an exception e terminates computation and passes e to the next higher level.
- **Example 6.28 (Division by Zero).** The Python interpreter reports unhandled exceptions.
- Exceptions are first class citizens in Python, in particular they
  - are classified by their classes in a hierarchy.
  - exception classes can be defined by the user (they inherit from the Exception class)
  - can be raised when an abnormal condition appears
  - can be handled in a **try**/**except** block (there can be multiple)

  try:
  $\langle\langle$tentativecomputation$\rangle\rangle$
  except : $\langle\langle$err$\rangle\rangle_1$, ..., $\langle\langle$err$\rangle\rangle_n$:
  $\langle\langle$errorhandling$\rangle\rangle$
  finally :
  $\langle\langle$cleanup$\rangle\rangle$

Playing it Safe with Databases

- **Observation 6.29.** *Things can go wrong when connecting to a database!* (*e.g. missing file*)
- **Idea:** Raise exceptions and handle them.
- **Example 6.30.** we encapsulate a **try/except** block into a function for convenience

```
import sqlite3
from sqlite3 import Error
def sql_connection():
    try:
        db = sqlite3.connect(':memory:')
        print("Connection is established: Database is created in memory")
    except Error :
        print(Error)
    finally:
        db.close()
```

The sqlite3 package provides its own exceptions, which we import separately. Other errors can be handled in additional **except** clauses.

9.7 Querying and Views in SQL

SQL Querying: The SELECT Statement

- SQL uses the **SELECT** instruction for retrieving data from a database.
- **SELECT** $\langle\langle \text{columns} \rangle\rangle$ **FROM** $\langle\langle \text{table} \rangle\rangle$ returns all records from $\langle\langle \text{table} \rangle\rangle$ restricted to the fields from $\langle\langle \text{columns} \rangle\rangle$.
- **Definition 7.1.** We call a **SELECT** instruction a query.

SQL Querying: The SELECT Statement

- SQL uses the **SELECT** instruction for retrieving data from a database.
- **SELECT** ⟪columns⟫ **FROM** ⟪table⟫ returns all records from ⟪table⟫ restricted to the fields from ⟪columns⟫.
- **Definition 7.5.** We call a **SELECT** instruction a query.
- **Example 7.6.** **SELECT** Title, YOP **FROM** Books;
- **SELECT DISTINCT** removes duplicate values
- **SELECT** $*$ **FROM** ⟪table⟫ returns all records from ⟪table⟫.

SQL Querying: The SELECT Statement

- SQL uses the **SELECT** instruction for retrieving data from a database.
- **SELECT** ⟪columns⟫ **FROM** ⟪table⟫ returns all records from ⟪table⟫ restricted to the fields from ⟪columns⟫.
- **Definition 7.9.** We call a **SELECT** instruction a query.
- **Example 7.10.** **SELECT** Title, YOP **FROM** Books;
- **SELECT DISTINCT** removes duplicate values
- **SELECT** * **FROM** ⟪table⟫ returns all records from ⟪table⟫.
- **SELECT** ⟪columns⟫ **FROM** ⟪table⟫ **WHERE** ⟪cond⟫ returns all records that match condition ⟪cond⟫
- **Example 7.11.** **SELECT** FirstN, LastN **FROM** Books **WHERE** YOP = 1995;

```
Willa|Cather
Ernest|Hemingway
Thomas|Wolfe
```

SQL Querying: The SELECT Statement

- SQL uses the **SELECT** instruction for retrieving data from a database.
- **SELECT** ⟪columns⟫ **FROM** ⟪table⟫ returns all records from ⟪table⟫ restricted to the fields from ⟪columns⟫.
- **Definition 7.13.** We call a **SELECT** instruction a query.
- **Example 7.14.** **SELECT** Title, YOP **FROM** Books;
- **SELECT DISTINCT** removes duplicate values
- **SELECT** $*$ **FROM** ⟪table⟫ returns all records from ⟪table⟫.
- **SELECT** ⟪columns⟫ **FROM** ⟪table⟫ **WHERE** ⟪cond⟫ returns all records that match condition ⟪cond⟫
- **Example 7.15.** **SELECT** FirstN, LastN **FROM** Books **WHERE** YOP = 1995;
- **SELECT** ⟪columns⟫ **FROM** ⟪table⟫ **ORDER BY** ⟪colums⟫ orders the results by ⟪columns⟫

SQL Querying: The SELECT Statement

- SQL uses the **SELECT** instruction for retrieving data from a database.
- **SELECT** $\langle\langle$columns$\rangle\rangle$ **FROM** $\langle\langle$table$\rangle\rangle$ returns all records from $\langle\langle$table$\rangle\rangle$ restricted to the fields from $\langle\langle$columns$\rangle\rangle$.
- **Definition 7.17.** We call a **SELECT** instruction a query.
- **Example 7.18.** **SELECT** Title, YOP **FROM** Books;
- **SELECT DISTINCT** removes duplicate values
- **SELECT** $*$ **FROM** $\langle\langle$table$\rangle\rangle$ returns all records from $\langle\langle$table$\rangle\rangle$.
- **SELECT** $\langle\langle$columns$\rangle\rangle$ **FROM** $\langle\langle$table$\rangle\rangle$ **WHERE** $\langle\langle$cond$\rangle\rangle$ returns all records that match condition $\langle\langle$cond$\rangle\rangle$
- **Example 7.19.** **SELECT** FirstN, LastN **FROM** Books **WHERE** YOP = 1995;
- **SELECT** $\langle\langle$columns$\rangle\rangle$ **FROM** $\langle\langle$table$\rangle\rangle$ **ORDER BY** $\langle\langle$colums$\rangle\rangle$ orders the results by $\langle\langle$columns$\rangle\rangle$
- **Example 7.20.** Ordering can be ascending (**ASC**) or descending (**DESC**)
  SELECT FirstN, LastN **FROM** Books **ORDER BY** LastN **ASC**, YOP **DESC**;

Joining Tables in Queries

- **Problem:** We can query single tables, how cross-table queries? E.g. in

Authors
AuthorID
Last Name
First Name
Birth Date
Death Date

Works
Title
PubDate
AuthorID
PublisherID

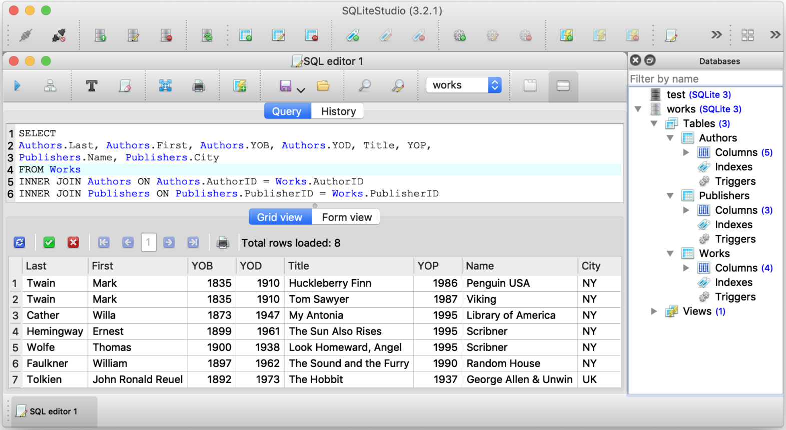
Publishers
PublisherID
Name
City

- **Idea:** Virtually join tables for the query! (as if we had the large books table)
- **Definition 7.21.** A table join (or simply join) is a means for combining columns from one (self join) or more tables by using values common to each.
- **Example 7.22.** Joining all three tables from 4.2.

```
SELECT
  Authors.LastN, Authors.FirstN, Authors.YOB, Authors.YOD,
  Title, YOP, Publishers.Name, Publishers.City
FROM
  Works
INNER JOIN Authors ON Authors.AuthorID = Works.AuthorID
INNER JOIN Publishers ON Publishers.PublisherID = Works.PublisherID
```

Joining Tables in Queries (Result)

- **Example 7.23.**



Database Views: Persisting Queries

- **Observation:** Via the join in 7.22, the Works table queries like the original Books table.
- **Wouldn't it be nice** If we could also insert/update into that?
- **Definition 7.24.** A database view (or simply view) is a virtual table based on the result set of a query. A view contains rows and columns, just like a real table. The field in a view are fields from one or more real tables in the database.
- *Remark 7.25.* In many RDBMS we can even insert, delete, and update records in a view, just as in any other table of the database.
  The RDBMS achieves this by automatically translating any change to the view into a set of changes to the underlying physical tables.
- ⚠ but not in SQLite. (this is an omission due to simplicity)

Database Views: Persisting Queries (Books Example)

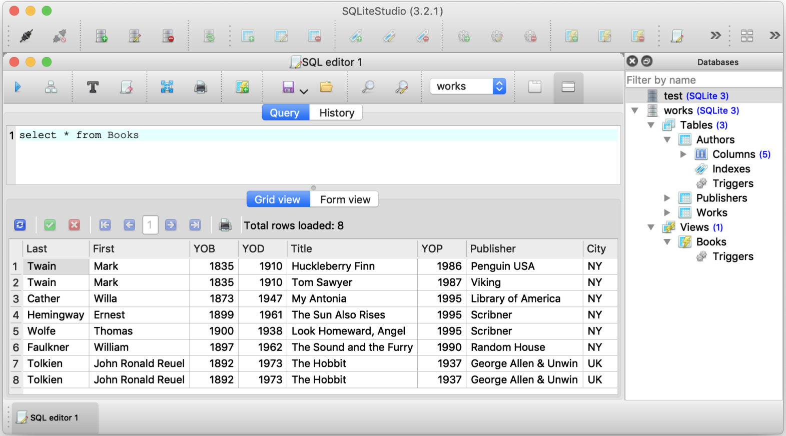
- **Example 7.26.** Use the query from 7.22 to define a view

```
CREATE VIEW Books AS
SELECT
  Authors.LastN AS LastN, Authors.FirstN AS FirstN,
  Authors.YOB AS YOB, Authors.YOD AS YOD,
  Title, YOP,
  Publishers.Name AS Publisher, Publishers.City AS City
FROM
  Works
INNER JOIN Authors ON Authors.AuthorID = Works.AuthorID
INNER JOIN Publishers ON Publishers.PublisherID = Works.PublisherID
```

Use AS clauses in SELECT to specify column names.

Database Views: Persisting Queries (Books Example)

▶ **Example 7.27.**



9.8 Querying via Python

Working with Cursors

- **Definition 8.1.** A cursor is a named object that encapsulates a set of query results in a (virtual) database table.
- To work with a cursor in sqlite3,
  - create a cursor object via the cursor method of your database object.
  - Open the cursor to establish the result set via its **execute** method
  - Fetch the data into local variables as needed from the cursor.
- The cursor class in sqlite3 provides additional methods:
  - fetchone(): return one row as an array/list
  - fetchall(): return all rows a list of lists.
  - fetchsome(⟪n⟫): return ⟪n⟫ rows a list of lists.
  - rowcount(): the number of rows in the cursor
- **Intuition:** Cursors allow programmers to repeatedly use a database query.

Extended Example: Listing Authors from the Books Table

- **Example 8.2.**

```
sql = 'SELECT␣FirstN,␣LastN,␣YOB␣FROM␣Books␣WHERE␣YOD␣<␣1950;'
cursor.execute(sql)
print ('There␣are␣',cursor.rowcount,'␣books,␣whose␣authors␣died␣before␣1950:\n')
for row in cursor.fetchall() :
    print (row[0],'␣␣',row[1], ';␣␣born␣',row[3],'\n')
print('That␣is␣all;␣if␣you␣want␣more,␣add␣more␣to␣the␣database!')
```

Inserting Multiple Records (Example)

- The cursor.executemany method takes an SQL instruction with parameters and a list of suitable tuples and executes them.
- **Example 8.3.** So the final form of insertion in 5.1 would be to define variable with a list of book tuples:

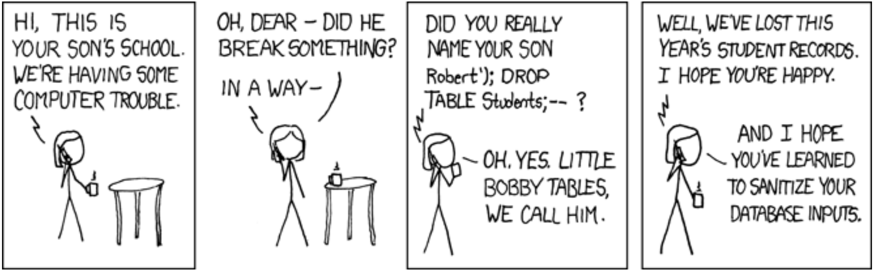
```
booklist = [
    ('Twain', 'Mark', 1835, 1910, 'Huckleberry␣Finn', 1986, 'Penguin␣USA', 'NY'),
    ('Twain', 'Mark', 1835, 1910, 'Tom␣Sawyer', 1987, 'Viking', 'NY'),
    ('Cather', 'Willa', 1873, 1947, 'My␣Antonia', 1995, 'Library␣of␣America', 'NY'),
    ('Hemingway', 'Ernest', 1899, 1961, 'The␣Sun␣Also␣Rises', 1995, 'Scribner', 'NY'),
    ('Wolfe', 'Thomas', 1900, 1938, 'Look␣Homeward,␣Angel', 1995, 'Scribner', 'NY'),
    ('Faulkner', 'William', 1897, 1962, 'The␣Sound␣and␣the␣Furry', 1990, 'Random␣House␣', 'N
    ('Tolkien', 'John␣Ronald␣Reuel', 1892, 1973, 'The␣Hobbit', 1937,'George␣Allen␣ Unwin', 'UK
```

and then insert it via a call of cursor.executemany:

```
cursor.executemany('INSERT␣INTO␣Books␣VALUES␣(?,?,?,?,?,?,?,?)',booklist)
```

Beware of the Python/SQLite Interaction

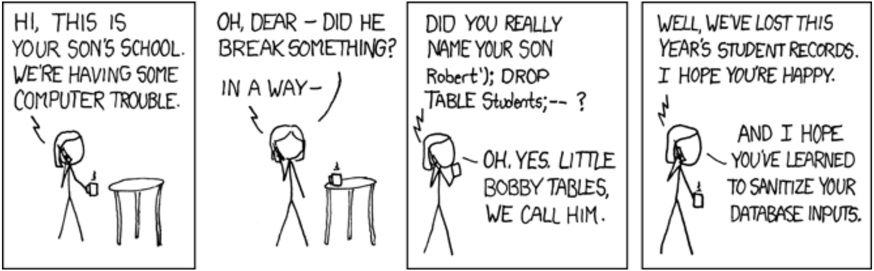
- **What have we learned?:** At least you now understand the following web comic: (https://xkcd.com/327/)



- **Definition 8.4.** We call this an SQL injection attack.

Beware of the Python/SQLite Interaction

- **What have we learned?:** At least you now understand the following web comic: (https://xkcd.com/327/)



- **Definition 8.5.** We call this an SQL injection attack.
- **Hint:** Imagine a web application where you add student names for enrolment.

```
name = input("Please␣enter␣student␣name:␣")
cursor.execute(f"INSERT␣INTO␣Students␣VALUES␣(...␣,{Name},␣...);")
```

For the input Robert');␣DROP␣TABLE␣Students; this has a Python line generates and executes the SQL instructions

```
INSERT INTO Students VALUES (..., 'Robert'); DROP TABLE Students;
```

SQLite3 Parameter Substitution

- **Observation 8.6.** *We often need variables as parameters in* cursor.execute*.*
- **Example 8.7.** In 8.2 we can ask the user for a year.
- **The python way** would be to use f strings

```
year = input('Books, whose author died before what year?')
sql = f'SELECT FirstN, LastN, YOB FROM Books WHERE YOD < {year}'
cursor.execute(sql) # ⚠ never use f−strings here −−> insecure
```

  But this leads to vulnerability by SQL injection attacks. (⇝ Bobby Tables)
- **Definition 8.8.** sqlite3 supplies a parameter substitution that SQL sanitizes parameters (removes problematic SQL instructions).
- **The sqlite3 way** uses parameter substitution (multiple ? possible ⇝ tuple)

```
year = input('Books, whose author died before')
select = 'SELECT Title FROM Books WHERE YOD < ?'
cursor.execute(select,(year,))
```

  or in the "named style" ⇝ order-independent (argument is a dictionary)

```
century = input('Century of the books?')
select = 'SELECT Title, YOP FROM Books WHERE YOP <= :start AND YOP > :end'
datadict = {'start': (century − 1) ∗ 100, 'end': century ∗ 100}
cursor.execute(select,datadict)
```

9.9 Real-Life Input/Output: XML and JSON

Filling a DB from via XML (Specification)

- **Idea:** We want to make a database based web application for NYC museums.
- **Recall** the public catalog from Example 4.5.4 (Introduction to XML) in the IWGS lecture notes, the XML file is online at `https://data.cityofnewyork.us/download/kcrm-j9hh/application/xml`

```xml
<?xml version="1.0" encoding="UTF-8"?>
<museums>
  <museum>
    <name>American Folk Art Museum</name>
    <phone>212-265-1040</phone>
    <address>45 W. 53rd St. (at Fifth Ave.)</address>
    <closing>Closed: Monday</closing>
    <rates>admission: $9; seniors/students, $7; under 12, free</rates>
    <specials>
      Pay-what-you-wish: Friday after 5:30pm;
      refreshments and music available
    </specials>
  </museum>
  <museum>
    <name>American Museum of Natural History</name>
    <phone>212-769-5200</phone>
    <address>Central Park West (at W. 79th St.)</address>
    <closing>Closed: Thanksgiving Day and Christmas Day</closing>
```

Filling a DB from via XML (Specification)

- **Idea:** We want to make a database based web application for NYC museums.
- **Recall** the public catalog from Example 4.5.4 (Introduction to XML) in the IWGS lecture notes, the XML file is online at `https://data.cityofnewyork.us/download/kcrm-j9hh/application/xml`
- **Idea:** We need Python program that
  - provides a SQLite database with a table 'museum' with columns 'name', 'phone', . . . , 'specials' of appropriate type
  - reads the XML file from the URL above and fills the table.
- **Possible Enhancement:** Encapsulate the functionality into a function, then we could run this program each night and keep the database up to date.

Filling a DB from via XML (Implementation)

- **Libraries:** urllib [UL] to retrieve the file and lxml [LXML] to parse it.

```
from lxml import etree
from urllib.request import urlopen
url = 'https://data.cityofnewyork.us/download/kcrm-j9hh/application/xml'
document = urlopen(url).read()
tree = etree.fromstring(document)
```

We now have a (large) XML tree in tree!

Filling a DB from via XML (Implementation)

- **Libraries:** urllib [UL] to retrieve the file and lxml [LXML] to parse it.
- Collect all the XML tags in all the museums (for the column names)

```
tags = []
for museum in tree:
    for info in museum:
        if info.tag not in tags:
            tags.append(info.tag)
```

- We create the SQLite database as discussed in slide 238.

Filling a DB from via XML (Implementation)

- **Libraries:** urllib [UL] to retrieve the file and lxml [LXML] to parse it.
- Collect all the XML tags in all the museums (for the column names)
- We create the SQLite database as discussed in slide 238.
- Then we assemble a table specification in a string columns:

```python
columns = ""
for cn in tags:
    # All columns have their name and type TEXT
    columns += f", {cn} TEXT"
```

Filling a DB from via XML (Implementation)

- **Libraries:** urllib [UL] to retrieve the file and lxml [LXML] to parse it.
- Collect all the XML tags in all the museums (for the column names)
- We create the SQLite database as discussed in slide 238.
- Then we assemble a table specification in a string columns:
- Create the Museums table from the specification in columns

```
cursor.execute("DROP␣TABLE␣IF␣EXISTS␣Museums;")
cursor.execute(f"""CREATE TABLE Museums
                    (Id INTEGER PRIMARY KEY {columns});""")
```

Filling a DB from via XML (Implementation)

- **Libraries:** urllib [UL] to retrieve the file and lxml [LXML] to parse it.
- Collect all the XML tags in all the museums (for the column names)
- We create the SQLite database as discussed in slide 238.
- Then we assemble a table specification in a string columns:
- Create the Museums table from the specification in columns
- Now the most important part: We fill the database

```
for museum in tree:
    # Find and sanitise the contents of all child nodes of this museum.
    values = []
    for tag in tags:
        if museum.find(tag) != None:
            values.append(str(museum.find(tag).text).strip())
        else:
            values.append("-")

    # Insert the data for this museum into the database.
    cols = str(tuple(tags))

    # We need a tuple of one ? for each column.
    vals = "(" + ("?, " * len(tags))[:-2] + ")"

    insert = f"INSERT INTO Museums {cols} VALUES {vals}"
    cursor.execute(insert, tuple(values))
```

Filling a DB from via XML (Implementation)

- **Libraries:** urllib [UL] to retrieve the file and lxml [LXML] to parse it.
- Collect all the XML tags in all the museums (for the column names)
- We create the SQLite database as discussed in slide 238.
- Then we assemble a table specification in a string columns:
- Create the Museums table from the specification in columns
- Now the most important part: We fill the database
- We finalize the transaction as discussed in slide 238.

The complete code in one block – a mere 51 lines

```
import sqlite3
from lxml import etree
from urllib.request import urlopen

# Download the XML file and Parse it
url = 'https://data.cityofnewyork.us/download/kcrm-j9hh/application/xml'
document = urlopen(url).read()
tree = etree.fromstring(document)

# First run-through of the XML: Collect the info types there,
tags = []
for museum in tree:
    for info in museum:
        if info.tag not in tags:
            tags.append(info.tag)

# Next, create database accordingly. First assemble a columns string.
columns = ""
for cn in tags:
    # All columns have their name and type TEXT
```

JSON — JavaScript Object Notation

- **Definition 9.1.** JSON (JavaScript Object Notation) is an open standard file format for interchange of structured data. JSON uses human readable text to store and transmit data objects consisting of attribute–value pairs and sequences.
- ⚠ JSON is very flexible, there need not be a regularizing schema.

JSON — JavaScript Object Notation

- **Definition 9.3.** JSON (JavaScript Object Notation) is an open standard file format for interchange of structured data. JSON uses human readable text to store and transmit data objects consisting of attribute–value pairs and sequences.
- ⚠ JSON is very flexible, there need not be a regularizing schema.
- **Intuition:** JSON is for JavaScript as (nested) dictionaries are for Python.
  - The browser can directly read JSON and use it via JavaScript.
  - $\leadsto$ AJAX $\widehat{=}$ JavaScript can query the backend for JSON data to update parts of the DOM. (lightweight interaction)
- **Consequence:**
  JSON is the dominant interchange format for web applications.

JSON — JavaScript Object Notation

- **Definition 9.5.** JSON (JavaScript Object Notation) is an open standard file format for interchange of structured data. JSON uses human readable text to store and transmit data objects consisting of attribute–value pairs and sequences.
- ⚠ JSON is very flexible, there need not be a regularizing schema.
- **Intuition:** JSON is for JavaScript as (nested) dictionaries are for Python.
  - The browser can directly read JSON and use it via JavaScript.
  - $\leadsto$ AJAX $\widehat{=}$ JavaScript can query the backend for JSON data to update parts of the DOM. (lightweight interaction)
- **Consequence:**
  JSON is the dominant interchange format for web applications.
- **Another Intuition:** JSON objects are like database records, but less rigid.
- **Idea:** Build a special JSON database. (JSON I/O; efficient storage)
- **Definition 9.6.** mongoDB is the most popular NoSQL database system. (no SQL inside)

Dealing with JSON in Python

- ⚠ Even though JSON concepts and syntax are similar to Python dictionaries, there are (subtle) differences.
- **Concretely:** Python allows more data types in dictionaries, e.g.

Python	JSON equivalent
`True`	`true`
`False`	`false`
`float`	`Number`
`int`	`Number`
`None`	`null`
`dict`	`Object`
`list`	`Array`
`tuple`	`Array`

- But these differences are systematic and can be overcome via the json library [JS].
  - json.dumps(⟪dict⟫) takes a Python dictionary dict, produces a JSON string.
  - json.loads(⟪json⟫) takes a JSON string json, produces a Python dictionary.

  There are many ways to control the output (pretty-printing), see [JS].

JSON Output for the NYC Museums DB

- **Libraries:** json for JSON [JS] and sqlite3 for the database.

```
import json
import sqlite3
```

JSON Output for the NYC Museums DB

- **Libraries:** json for JSON [JS] and sqlite3 for the database.
- Connect to the SQLite database as usual and query the database for everything

```
db = sqlite3.connect("./museums.sqlite")
cursor = db.cursor()
cursor.execute("SELECT * FROM Museums;")
```

JSON Output for the NYC Museums DB

- **Libraries:** json for JSON [JS] and sqlite3 for the database.
- Connect to the SQLite database as usual and query the database for everything
- Initialize a dictionary and the list of Museums column names

```
data = {}
data['museums'] = []
columns = ['name', 'phone', 'address', 'closing', 'rates', 'specials']
```

JSON Output for the NYC Museums DB

- **Libraries:** json for JSON [JS] and sqlite3 for the database.
- Connect to the SQLite database as usual and query the database for everything
- Initialize a dictionary and the list of Museums column names
- For each of the rows in the Museums table build a row dictionary

```
for row in cursor.fetchall():
    # Generate a dictionary with columns as keys and entrys as values.
    rowdict = { columns[n] : row[n] for n in range(6) }

    # Add that dictionary to the JSON data structure.
    data['museums'].append(rowdict)
```

JSON Output for the NYC Museums DB

- **Libraries:** json for JSON [JS] and sqlite3 for the database.
- Connect to the SQLite database as usual and query the database for everything
- Initialize a dictionary and the list of Museums column names
- For each of the rows in the Museums table build a row dictionary
- Dump the data dictionary as JSON into a file

```
with open('museums.json', 'w') as outfile:
    json.dump(data, outfile)
```

- Close the database as usual.

JSON Output for the NYC Museums DB I

```python
import json
import sqlite3

# Connect to database and query database for everything.
db = sqlite3.connect("./museums.sqlite")
cursor = db.cursor()
cursor.execute("SELECT * FROM Museums;")

# Setup soon-to-be-JSON dictionary and the necessary columns
data = {}
data['museums'] = []
columns = ['name', 'phone', 'address', 'closing', 'rates', 'specials']

# For every row in the result, do the following:
for row in cursor.fetchall():
    # Generate a dictionary with columns as keys and entrys as values.
    rowdict = { columns[n] : row[n] for n in range(6) }

    # Add that dictionary to the JSON data structure.
    data['museums'].append(rowdict)

# Write collected JSON data to file.
with open('museums.json', 'w') as outfile:
```

JSON Output for the NYC Museums DB II

```
    json.dump(data, outfile)

# Close database
db.close()
```

JSON Example (NYC Museums)

- **Example 9.7.** The NYC museums data from Example 4.5.4 (Introduction to XML) in the IWGS lecture notes as JSON:
  We represent the data as a “sequence” of (nested) “dictionaries”

```
[
    {"name": "American Folk Art Museum",
     "phone": "212-265-1040",
     "address": "45 W. 53rd St. (at Fifth Ave.)",
     "closing": "Closed: Monday",
     "rates": {
         "admission": "$9",
         "seniors/students": "$7",
         "under 12": "free",
     }
     "specials": "Pay-what-you-wish: Friday after 5:30pm;
                  refreshments and music available"
    }
    {"name": "American Museum of Natural History",
     "phone": "212-769-5200",
     "address": "Central Park West (at W. 79th St.)"
     "closing": "Closed: Thanksgiving Day and Christmas Day"
     "rates": {
         "admission": "$16",
         "seniors/students": "$12",
         "kids 2-12": "$9",
         "under 2": "free"
```

Chapter 10
Project: A Web GUI for a Books Database

10.1 A Basic Web Application

Building a full Web Application with Database Backend

- **Observation 1.1.** *With the technology in 5 (Web Applications) in the IWGS lecture notes and we can build a full web application in less than*
  - *100 lines of Python code and* (*back-end/routes*)
  - *less than 70 lines of HTML template files.* (*front end*)
- **Functionality:** Manage a database of books, in particular: (e.g. your library at home)
  - add a new book to the database
  - delete a book from the database
  - update (i.e. change) an existing book
- The source is at `https://gl.mathhub.info/MiKoMH/IWGS/blob/master/source/booksapp/code/books-app.py`.

The Books Application: Setup

- We have already seen how to set up the database in slide 250.

```
import sqlite3
from sqlite3 import Error
from bottle import route, run, debug, template, request, get, post

# our database file
database = "books.db"
db = sqlite3.connect(database)
```

- But we want to receive result rows as dictionaries, not as tuples, so we add

```
db.row_factory = sqlite3.Row
```

The Books Application: Setup

- We have already seen how to set up the database in slide 250.

```
import sqlite3
from sqlite3 import Error
from bottle import route, run, debug, template, request, get, post

# our database file
database = "books.db"
db = sqlite3.connect(database)
```

- But we want to receive result rows as dictionaries, not as tuples, so we add

```
db.row_factory = sqlite3.Row
```

- We give ourselves a cursor to work with

```
cursor = db.cursor()
```

The Books Application: Setup

- We have already seen how to set up the database in slide 250.

```
import sqlite3
from sqlite3 import Error
from bottle import route, run, debug, template, request, get, post

# our database file
database = "books.db"
db = sqlite3.connect(database)
```

- But we want to receive result rows as dictionaries, not as tuples, so we add

```
db.row_factory = sqlite3.Row
```

- We give ourselves a cursor to work with

```
cursor = db.cursor()
```

- We start the bottle server

```
run(host='localhost', port=8080, debug=True)
```

- And of course, we eventually commit and close the database in the end

```
db.commit()
db.close()
```

The Books Application: Backend

- We specify the database schema and create the Books table

```
bookstable = """
CREATE TABLE IF NOT EXISTS Books (
    Last varchar(128), First varchar(128),
    YOB int, YOD int, Title varchar(255), YOP int,
    Publisher varchar(128), City varchar(128)
  );
  """
cursor.execute(bookstable)
```

The Books Application: Books to Play With

- Data about books as a Python list of 8-tuples:

```
initialbooklist = [
    ('Twain', 'Mark', 1835, 1910, 'Huckleberry␣Finn', 1986, 'Penguin␣USA', 'NY'),
    ('Twain', 'Mark', 1835, 1910, 'Tom␣Sawyer', 1987, 'Viking', 'NY'),
    ('Cather', 'Willa', 1873, 1947, 'My␣Antonia', 1995, 'Library␣of␣America', 'NY'),
    ('Hemingway', 'Ernest', 1899, 1961, 'The␣Sun␣Also␣Rises', 1995, 'Scribner', 'NY'),
    ('Wolfe', 'Thomas', 1900, 1938, 'Look␣Homeward,␣Angel', 1995, 'Scribner', 'NY'),
    ('Faulkner', 'William', 1897, 1962, 'The␣Sound␣and␣the␣Furry', 1990, 'Random␣House␣', 'N
    ('Tolkien', 'John␣Ronald␣Reuel', 1892, 1973, 'The␣Hobbit', 1937,'George␣Allen␣ Unwin', 'UK
```

The Books Application: Books to Play With

- Data about books as a Python list of 8-tuples:

```
initialbooklist = [
    ('Twain', 'Mark', 1835, 1910, 'Huckleberry␣Finn', 1986, 'Penguin␣USA', 'NY'),
    ('Twain', 'Mark', 1835, 1910, 'Tom␣Sawyer', 1987, 'Viking', 'NY'),
    ('Cather', 'Willa', 1873, 1947, 'My␣Antonia', 1995, 'Library␣of␣America', 'NY'),
    ('Hemingway', 'Ernest', 1899, 1961, 'The␣Sun␣Also␣Rises', 1995, 'Scribner', 'NY'),
    ('Wolfe', 'Thomas', 1900, 1938, 'Look␣Homeward,␣Angel', 1995, 'Scribner', 'NY'),
    ('Faulkner', 'William', 1897, 1962, 'The␣Sound␣and␣the␣Furry', 1990, 'Random␣House␣', 'N
    ('Tolkien', 'John␣Ronald␣Reuel', 1892, 1973, 'The␣Hobbit', 1937,'George␣Allen␣ Unwin', 'UK
```

- If the Books table is empty, we fill it with the tuples in initialbooklist:

```
row = cursor.execute('SELECT␣*␣FROM␣Books␣LIMIT␣1').fetchall()
if not row:
    cursor.executemany('INSERT␣INTO␣Books␣VALUES␣(?,?,?,?,?,?,?,?)',initialbooklist
```

- **Idea:** To find out if the table is empty (surprisingly clumsy)
  - we fetch a list with at most one row (LIMIT 1);
  - if Books is empty, row is the empty list which evaluates to false in a conditional.

The Books Application Routes: The Application Root

- We only need to add the bottle routes for the various sub pages.
- **The main page:** Listing the book records in the database

```
@route('/')
def books():
    query = 'SELECT␣rowid,Last,First,YOB,YOD,Title,YOP,Publisher,City␣FROM␣Books'
    cursor.execute(query)
    booklist = cursor.fetchall()
    return template('books',books=booklist,num=len(booklist),cols=cols)
```

- This uses the following templates: the first generates a table of books from the template file books.tpl

```
<p>There are {{num}} books in the database</p>
<table>
  % include('th.tpl', cols=cols)
  % for book in books : include('book.tpl',**book,cols=cols) end
  <tr><th><a href="/add"><button>add a book</button></a></th></tr>
</table>
```

The Books Application Root: Result

- Here is the page of the books application in its initial state.

There are 7 books in the database

Last	First	YOB	YOD	Title	YOP	Publisher	City	Action	
Twain	Mark	1835	1910	Huckleberry Finn	1986	Penguin USA	NY	edit	delete
Twain	Mark	1835	1910	Tom Sawyer	1987	Viking	NY	edit	delete
Cather	Willa	1873	1947	My Antonia	1995	Library of America	NY	edit	delete
Hemingway	Ernest	1800	1961	The Sun Also Rises	1995	Scribner	NY	edit	delete
Wolfe	Thomas	1900	1938	Look Homeward, Angel	1995	Scribner	NY	edit	delete
Faulkner	William	1897	1962	The Sound and the Furry	1990	Random House	NY	edit	delete
Tolkien	John Ronald Reuel	1892	1973	The Hobbit	1937	George Allen & Unwin	UK	edit	delete

add a book

The Books Application Root: More Templates

- **Recall:** The books.tpl template file

```
<p>There are {{num}} books in the database</p>
<table>
   % include('th.tpl', cols=cols)
   % for book in books : include('book.tpl',**book,cols=cols) end
   <tr><th><a href="/add"><button>add a book</button></a></th></tr>
</table>
```

  that generates this result via the following two templates:

- It inserts the table header via th.tpl:

```
% for col in cols:
  <th>{{col}}</th>
% end
<th rowspan="2">Action</th>
```

- and iterates over the list of books, using the template file book.tpl:

```
<tr>
  <td>{{Last}}</td><td>{{First}}</td><td>{{YOB}}</td><td>{{YOD}}</td>
  <td>{{Title}}</td><td>{{YOP}}</td><td>{{Publisher}}</td><td>{{City}}</td>
  <td><a href="/edit/{{rowid}}"><button>edit</button></a></td>
  <td><a href="/delete/{{rowid}}"><button >delete</button></a></td>
</tr>
```

- **Row Id Trick:** Note the slightly subtle use of the rowid column in this template. It is (only) used in the two action buttons to specify which book to add/edit.

The Books Application Routes: Adding Book Records

- We add a route for adding a books record (for the add button)

```
@get('/add')
def add():
    return template('add',cols=cols)
```

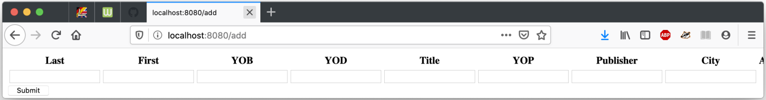
  Note that this is the route for the GET method on the path /add.

- This uses the template file add.tpl:

```
<form action="/add" method="post">
  <table>
    % include('th.tpl', cols=cols)
    <tr>
        % for td in cols:
        <td><input type="text" name="{{td}}"/></td>
        % end
    </tr>
  </table>
  <input type="submit" value="Submit"/>
</form>
```

The Books Application Routes: Adding Book Records

- The result is



- The action in the HTML form is to POST to the path /add. Thus we need POST route for /add as well:

```
@post('/add')
def addResponse():
    data = parseResponse()
    ins = '''INSERT INTO Books VALUES
            (:Last,:First,:YOB,:YOD,:Title,:YOP,:Publisher,:City)'''
    cursor.execute(ins,data)
    return template('response', data = data, cols=cols,
                                    rowid = cursor.lastrowid,
                                    text = 'New book record received')
```

Note the use of sqlite3 parameter substitution in addResponse!

The Books Application Routes: Adding Book Records

- This uses the function parseResponse, which we will reuse later.

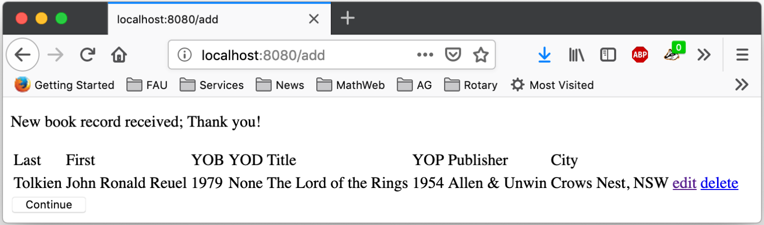
```
def parseResponse ():
    data = {'Last': request.forms.get('Last'),
            'First': request.forms.get('First'),
            'YOB': request.forms.get('YOB'),
            'YOD': request.forms.get('YOD'),
            'Title': request.forms.get('Title'),
            'YOP': request.forms.get('YOP'),
             'Publisher': request.forms.get('Publisher'),
            'City': request.forms.get('City')}
    return data
```

- and the template repsone.tpl:

```
<form action='/'>
  <p>{{text}}; Thank you!</p>
  <table>
    % include('th.tpl',cols=cols)
    % include('book.tpl',**data,cols=cols)
  </table>
  <input type="submit" value="Continue"/>
</form>
```

The Books Application Routes: Adding Book Records

- Here is the result after filling in Tolkien's "*Lord of the Rings*":



The Books Application Routes: Deleting Book Records

- We add a route for deleting book records (for the delete button)

```
@get('/delete/<id:int>')
def delete(id):
    cursor.execute('DELETE FROM Books WHERE rowid = ?',(id,))
    return template('delete')
```

  Note that we have a dynamic route here: We use the named wildcard <id:int> to obtain the rowid of the record to be deleted.

- The template file delete.tpl does the obvious:

```
<form action='/'>
  <p>Book record deleted; Thank you!</p>
  <input type="submit" value="Continue"/>
</form>
```

The Books Application Routes: Editing Book Records

- **Idea:** Combine techniques from the add and delete routes

```
@get('/edit/<id:int>')
def edit(id):
    cursor.execute('SELECT * FROM Books WHERE rowid = ?',(id,))
    return template('edit',cursor.fetchone(),id = id,cols=cols)

@post('/edit/<id:int>')
def editResponse(id):
    data = parseResponse()
    up = """UPDATE Books
          SET Last = :Last, First = :First, YOB = :YOB, YOD = :YOD,
              Title = :Title, YOP = :YOP, Publisher = :Publisher,
              City = :City
          WHERE rowid = :rowid"""
    data.update({'rowid': id})
    cursor.execute(up,data)
    return template('response',data=data,text='Updated book record',cols=cols)
```

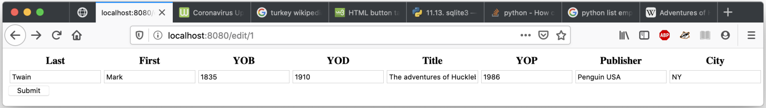
Books Application Routes: Editing Book Records (cont.)

- The template file edit.tpl is similar to add.tpl above, but pre-fills the input fields with the database record values.

```
<form action="/edit/{{id}}" method="post">
  <table>
    % include('th.tpl', cols=cols)
    <tr>
      <td><input type="text" name="Last" value="{{Last}}"/></td>
      <td><input type="text" name="First" value="{{First}}"/></td>
      <td><input type="text" name="YOB" value="{{YOB}}"/></td>
      <td><input type="text" name="YOD" value="{{YOD}}"/></td>
      <td><input type="text" name="Title" value="{{Title}}"/></td>
      <td><input type="text" name="YOP" value="{{YOP}}"/></td>
      <td><input type="text" name="Publisher" value="{{Publisher}}"/></td>
      <td><input type="text" name="City" value="{{City}}"/></td>
      <td><input type="submit" value="Submit"/></td>
    </tr>
  </table>
</form>
```

Books Application Routes: Editing Book Records (cont.)

- The result is



- Again, we use the template response.tpl, which we fill with a different message.

10.2 Access Control and Management

Access Control and Management

- **Problem:** Anyone can write, edit, and delete records from the books database.
- **Solution:** Implement a password-based log in procedure and restrict write/edit/delete access to logged-in agents.
- Let's fix some terminology before we continue

Access Control and Management

- **Problem:** Anyone can write, edit, and delete records from the books database.
- **Solution:** Implement a password-based log in procedure and restrict write/edit/delete access to logged-in agents.
- Let's fix some terminology before we continue
- **Definition 2.3.** Access control is the selective restriction of access to a resource, access management describes the corresponding process.
- Access management usually comprises both authentication and authorization.
- **Definition 2.4.** Authorization refers to a set of rules that determine who is allowed to do what with a collection of resources.

Access Control and Management

- **Problem:** Anyone can write, edit, and delete records from the books database.
- **Solution:** Implement a password-based log in procedure and restrict write/edit/delete access to logged-in agents.
- Let's fix some terminology before we continue
- **Definition 2.5.** Access control is the selective restriction of access to a resource, access management describes the corresponding process.
- Access management usually comprises both authentication and authorization.
- **Definition 2.6.** Authorization refers to a set of rules that determine who is allowed to do what with a collection of resources.
- **For our books application** we need four things
  1. a browser interaction to query the user for username and password
  2. a way to transport them to the web application program
  3. a method for checking the username/password (authentication)
  4. a way the specify who can do what. (authorization)

  Realization: 1./2. via HTTP, 4. via bottle basic auth, implement 3. directly.

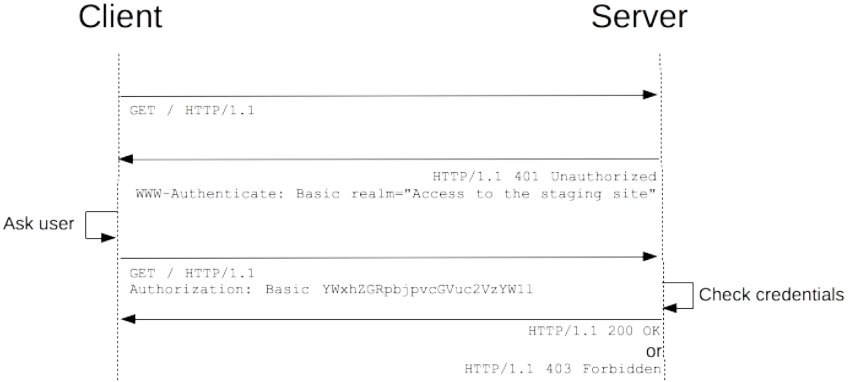
HTTP Basic Authentication

- **Recall** that HTTP is a plain text protocol that passes around headers like this

```
GET /docs/index.html HTTP/1.1
Host: www.nowhere123.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
(blank line)
```

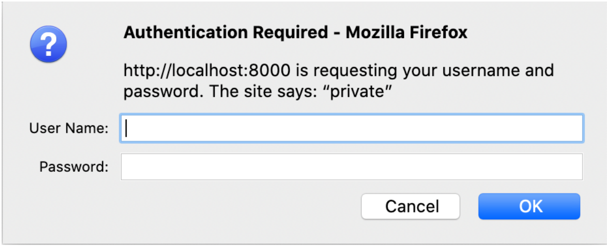
HTTP Basic Authentication

- **Recall** that HTTP is a plain text protocol that passes around headers like this
- **Idea:** For authentication extend the HTTP headers with support for username/password pairs.
- **Definition 2.8.** HTTP basic authentication introduces a HTTP header Authorization for base64 encoded pairs ⟪username⟫:⟪password⟫ and a couple of challenge/response messages.



HTTP Basic Authentication

- **Recall** that HTTP is a plain text protocol that passes around headers like this
- **Idea:** For authentication extend the HTTP headers with support for username/password pairs.
- **Definition 2.9.** HTTP basic authentication introduces a HTTP header Authorization for base64 encoded pairs $\langle\langle\text{username}\rangle\rangle$:$\langle\langle\text{password}\rangle\rangle$ and a couple of challenge/response messages.



HTTP Basic Authentication

- **Recall** that HTTP is a plain text protocol that passes around headers like this
- **Idea:** For authentication extend the HTTP headers with support for username/password pairs.
- **Definition 2.10.** HTTP basic authentication introduces a HTTP header Authorization for base64 encoded pairs $\langle\langle \text{username} \rangle\rangle:\langle\langle \text{password} \rangle\rangle$ and a couple of challenge/response messages.
- **Problem:** Base64 is very easy to decode, so usernames and passwords are communicated in the clear (very unsafe)
- Passwords are "binary data" (think special characters), encoding just keeps them unchanged over the network. (no encryption)

Basic Auth in Bottle

- **Idea:** Support the server side of HTTP basic authentication in bottle web-apps.
- **Implementation:** New decorator @auth_basic(⟪function⟫) to mark a route as password-protected.
- **Usage:** Decorate every route we want to restrict access of with @auth_basic(⟪function⟫), where ⟪function⟫ is a function that takes two string arguments (user name and password) and returns a Boolean for the authorization decision.

Basic Auth in Bottle: Minimal Viable Example

- **Example 2.11.** A web application with restricted route.

```
from bottle import run, get, auth_basic

def check(user, password):
    return user == "miko" and password == "test"

@get("/")
@auth_basic(check)
def protected():
    return "Authorized access granted!"

run(host="localhost", port=8000)
```

- **Idea:** Mix restricted and open routes in a partially restricted application.
- **Extension:** Use different check functions for different levels of restriction (user roles)

HTTPS: HTTP over TLS

- **Definition 2.12.** Hypertext Transfer Protocol Secure (HTTPS) is an extension of the Hypertext Transfer Protocol (HTTP) for secure communication over a computer network. HTTPS achieves this by running HTTP over a TLS connection.
- **Consequences for Web Applications:** We can use HTTP as usual, except
  - we gain communication privacy and server authentication,
  - server and browser need to speak HTTPS, (most do)
  - the server needs a public key certificate and a private key.

HTTPS: HTTP over TLS

- **Definition 2.13.** Hypertext Transfer Protocol Secure (HTTPS) is an extension of the Hypertext Transfer Protocol (HTTP) for secure communication over a computer network. HTTPS achieves this by running HTTP over a TLS connection.
- **Consequences for Web Applications:** We can use HTTP as usual, except
  - we gain communication privacy and server authentication,
  - server and browser need to speak HTTPS, (most do)
  - the server needs a public key certificate and a private key.
- In bottle, we can just swap out the HTTP server to one that can do HTTPS:

```
run(host='localhost',port='8888',
    server='gunicorn',keyfile='key.pem',certfile='cert.pem')
```

install it first with pip install gunicorn.

HTTPS: HTTP over TLS

- **Definition 2.14.** Hypertext Transfer Protocol Secure (HTTPS) is an extension of the Hypertext Transfer Protocol (HTTP) for secure communication over a computer network. HTTPS achieves this by running HTTP over a TLS connection.
- **Consequences for Web Applications:** We can use HTTP as usual, except
  - we gain communication privacy and server authentication,
  - server and browser need to speak HTTPS, (most do)
  - the server needs a public key certificate and a private key.
- In bottle, we can just swap out the HTTP server to one that can do HTTPS:

```
run(host='localhost',port='8888',
    server='gunicorn',keyfile='key.pem',certfile='cert.pem')
```

  install it first with pip install gunicorn.
- **Problem:** Where to get the certificate file cert.pem and private key key.pem?

HTTPS: HTTP over TLS

- **Definition 2.15.** Hypertext Transfer Protocol Secure (HTTPS) is an extension of the Hypertext Transfer Protocol (HTTP) for secure communication over a computer network. HTTPS achieves this by running HTTP over a TLS connection.
- **Consequences for Web Applications:** We can use HTTP as usual, except
  - we gain communication privacy and server authentication,
  - server and browser need to speak HTTPS, (most do)
  - the server needs a public key certificate and a private key.
- In bottle, we can just swap out the HTTP server to one that can do HTTPS:

```
run(host='localhost',port='8888',
    server='gunicorn',keyfile='key.pem',certfile='cert.pem')
```

  install it first with pip install gunicorn.
- **Problem:** Where to get the certificate file cert.pem and private key key.pem?
- **One Solution:** Self-sign one, e.g. using `https://www.selfsignedcertificate.com/` (adapt file names)
- **Remaining Problem:** Your browser forces you to specify an exception for `https://localhost:8888` (probably OK for development)

Getting a Real TLS Certificate via Let's-Encrypt

- **Intuition:** HTTPS is the new "regular HTTP" on the web!
- **Observation 2.16.** *A self-signed certificate gives communication privacy but not authentication* $\leadsto$ *only you yourself vouch for the authenticity of the web site.*

Getting a Real TLS Certificate via Let's-Encrypt

- **Intuition:** HTTPS is the new "regular HTTP" on the web!
- **Observation 2.19.** *A self-signed certificate gives communication privacy but not authentication ⇜ only you yourself vouch for the authenticity of the web site.*
- **Definition 2.20.** In a public key infrastructure, the TLS certificate is issued by a certificate authority, an organization chartered to verify identity and issue TLS certificates.
- Commercial certificate authorities sell trust. (for a lot of money)
  They certify e.g. that the `https://bmw.com` is under control of BMW AG.

Getting a Real TLS Certificate via Let's-Encrypt

- **Intuition:** HTTPS is the new "regular HTTP" on the web!
- **Observation 2.22.** *A self-signed certificate gives communication privacy but not authentication ⇝ only you yourself vouch for the authenticity of the web site.*
- **Definition 2.23.** In a public key infrastructure, the TLS certificate is issued by a certificate authority, an organization chartered to verify identity and issue TLS certificates.
- Commercial certificate authorities sell trust. (for a lot of money)
  They certify e.g. that the `https://bmw.com` is under control of BMW AG.
- **Idea:** Finding out that you have control over a particular web site on the web can be automated, if you run a program on the server host.
- **Definition 2.24.** Let's Encrypt is a not for profit certificate authority that does this and issues free TLS certificates. (to encourage HTTPS adoption)

Getting a Real TLS Certificate via Let's-Encrypt

- **Intuition:** HTTPS is the new "regular HTTP" on the web!
- **Observation 2.25.** *A self-signed certificate gives communication privacy but not authentication ⇝ only you yourself vouch for the authenticity of the web site.*
- **Definition 2.26.** In a public key infrastructure, the TLS certificate is issued by a certificate authority, an organization chartered to verify identity and issue TLS certificates.
- Commercial certificate authorities sell trust. (for a lot of money)
  They certify e.g. that the `https://bmw.com` is under control of BMW AG.
- **Idea:** Finding out that you have control over a particular web site on the web can be automated, if you run a program on the server host.
- **Definition 2.27.** Let's Encrypt is a not for profit certificate authority that does this and issues free TLS certificates. (to encourage HTTPS adoption)
- **Concretely:** on a linux server you need two steps
  1. install certbot (usually via your package manager)
  2. then sudo /usr/local/bin/certbot certonly −−standalone will generate certs.

  Details at `https://letsencrypt.org`.
- **Success:** $\geq$ 1.000.000.000 TLS certificates, 200.000.000 sites since 2016

10.3 Asynchronous Loading in Modern Web Apps

AJAX for more responsive Web Pages

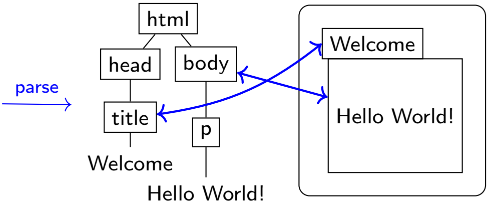
- **Definition 3.1.** Ajax, (also AJAX; short for "Asynchronous JavaScript and XML") is a set of client side techniques for creating asynchronous web applications.
- **Definition 3.2.** A process p is called asynchronous, iff the parent process (i.e. the one that spawned p) continues processing without waiting for p to terminate.
- **Intuition:** With Ajax, web applications can send and retrieve data from a server without interfering with the display and behaviour of the existing page.
- **Application:** By decoupling the data interchange layer from the presentation layer, Ajax allows web pages and, by extension, web applications, to change content dynamically without the need to reload the entire page.
- **Observation:** Almost all modern web application extensively utilize Ajax.
- **Note:** In practice, modern implementations commonly use JSON instead of XML.

Background: Rendering Pipeline in browsers

- **Observation:** The nested markup codes turn HTML documents into trees.
- **Definition 3.3.** The document object model (DOM) is a data structure for the HTML document tree together with a standardized set of access methods.
- **Rendering Pipeline:** Rendering a web page proceeds in three steps
  1. the browser receives a HTML document,
  2. parses it into an internal data structure, the DOM,
  3. which is then painted to the screen. (repaint whenever DOM changes)

HTML Document **DOM** **Browser**

```
<html>
  <head>
    <title>Welcome</title>
  </head>
  <body>
    <p>Hello World!</p>
  </body>
</html>
```



The DOM is notified of any user events (resizing, clicks, hover,. . .)

Example: Details on Request via AJAX

- **Idea:** Use Ajax in a web application for the books application
  - The start page just has a list of book titles, and
  - details are fetched by an Ajax request and presented in line.
- **Planning the Program:** We need a bottle server with
  1. a dynamic route that returns JSON-encoded data for a given book,
  2. a route for the main page that lists the book titles,
  3. stpl template files for list items with an Ajax request, and
  4. a JavaScript function that reads the JSON and inserts it into the DOM.

The finished product (initial state)

Books by Title

1. Tom Sawyer (show details)
2. My Antonia (show details)
3. The Sun Also Rises (show details)
4. Look Homeward, Angel (show details)
5. The Sound and the Furry (show details)
6. The Hobbit (show details)

The finished product (with details loaded)

Books by Title

1. Tom Sawyer

   Author: Mark Twain (1835 - 1910)

   Publisher: Viking, 1987

   (hide details)
2. My Antonia (show details)
3. The Sun Also Rises (show details)
4. Look Homeward, Angel (show details)
5. The Sound and the Furry (show details)
6. The Hobbit (show details)

The Routes (Serving HTML and JSON)

- After setting up the database and co, we have a standard route:

```
@route('/')
def books():
    cursor.execute('SELECT rowid, Title, YoP FROM Books')
    rv = cursor.fetchall()
    return template('titles', books=rv)
```

- JSON routes and APIs are very easy in bottle: we just return a dictionary.

```
@route('/json/<id:int>')
def book(id):
    cursor.execute(f'SELECT * FROM Books WHERE rowid={id}')
    row = cursor.fetchone() # Only one result, rowid is a primary key.
    return dict(zip(row.keys(), row)) # Pair up column names with values.
```

- **Dictionaries and JSON in Bottle:** Bottle automatically transforms Python dictionaries into JSON strings; sets the Content Type header to application/json.

The Basic Templates

- The template titles.tpl is also standard

```
<html>
% include('bookshead.tpl')
<body>
  <h1>Books by Title</h1>
  <ol>
    % for bk in books: include('title.tpl',Id=bk[0], title=bk[1]) end
  <ol>
</body>
</html>
```

- The template title.tpl presents a single book title

```
<li>
  <span class="booktitle">{{title}}</span>
  <span id="content{{Id}}"></span>
  <span class="interact" id="interact{{Id}}"
              onclick="load_details({{Id}})">(show details)</span>
</li>
```

  The empty span will be filled by an Ajax call later!

- The interesting things happen in bookshead.tpl (up next)

The Script load_details

- bookshead.tpl starts supplying JQuery and a JQuery templating library:

```
<script type="application/javascript"
    src="http://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
<script type="application/javascript"
    src="https://cdn.jsdelivr.net/gh/codepb/jquery−template@1.5.10/dist/jquery.loadTemplate.
```

- The main contribution of bookshead.tpl is the JQuery function load_details

```
async function load_details (numb) {
  /* Request Info via JSON, feed it to template, update "show␣details" span */
  await $.getJSON("/json/" + numb,
      function (data) {$("#content" + numb).loadTemplate($("#open"), data)});
```

which uses the JQuery Ajax call $.getJSON. This takes two arguments:

1. the URL for the HTTP GET request
2. a JavaScript function that is called if the GET request was successful.

The function (in argument 2) is then used to extend the result of $("#content"+ numb), i.e. that element in the DOM whose id attribute is content*i* where *i* is the value of the numb variable.

The Script load_details Continued

- We also use JQuery to change the onlick behaviour of the span element (from load_details to toggle_details, explained below) and the text contained therein.

```
  interact = $("#interact" + numb)

  /* change click behaviour of interaction span from show to toggle */
  interact.removeAttr('onclick');
  interact.attr('onClick', 'toggle_details(' + numb + ');');

  /* also change included text appropriately */
  interact.html("(hide␣details)");
}
```

The Script load_details Continued

- We also use JQuery to change the onlick behaviour of the span element (from load_details to toggle_details, explained below) and the text contained therein.
- Recall the structure of title.tpl: For every book we have a title, a content element that starts out empty and gets filled when load_details is called, and a clickable interaction element that triggers load_details.

```
<li>
  <span class="booktitle">{{title}}</span>
  <span id="content{{Id}}"></span>
  <span class="interact" id="interact{{Id}}"
              onclick="load_details({{Id}})">(show details)</span>
</li>
```

The Script load_details Continued

- We also use JQuery to change the onlick behaviour of the span element (from load_details to toggle_details, explained below) and the text contained therein.
- Recall the structure of title.tpl: For every book we have a title, a content element that starts out empty and gets filled when load_details is called, and a clickable interaction element that triggers load_details.
- The toggle_details-function used above does nothing but setting the content element to hidden or visible and changing the text of the interaction element.

```
function toggle_details (numb) {
  /* hide or show appropriate content element */

  content = $("#content" + numb);
  interact = $("#interact" + numb);

  if(content.css('display') == 'none') {
    content.show();
    interact.html("(hide details)");
  } else {
    content.hide();
    interact.html("(show details)");
  }
}
```

JQuery Template Processing

- **Recall:** We are still trying to understand
  $("#content" + numb).loadTemplate($("#open''),data)
  It extends the empty <span id=''content*i*> in title.tpl with a details table:

JQuery Template Processing

- **Recall:** We are still trying to understand
  $("#content" + numb).loadTemplate($("#open''),data)
  It extends the empty <span id=''content*i*> in title.tpl with a details table:
- The loadTemplate method takes two arguments
  1. a template; here the result of $(#open),i.e. the element in bookshead.tpl whose id attribute is open (note the type attribute that makes it HTML)

```
<script type="text/html" id="open">
  <table>
     <tr>
       <th>Author:</th>
       <td>
         <span data−content="First"></span> <span data−content="Last"></span>
          (<span data−content="YOB">−</span> − <span data−content="YOD"><
       </td>
     </tr>
     <tr>
        <th>Publisher:</th>
        <td><span data−content="Publisher"></span>, <span data−content="YOP"
    </tr>
  </table>
</script>
```

JQuery Template Processing

- **Recall:** We are still trying to understand
  $("#content" + numb).loadTemplate($("#open''),data)
  It extends the empty <span id=''content*i*> in title.tpl with a details table:
- The loadTemplate method takes two arguments
  1. a template; here the result of $(#open),i.e. the element in bookshead.tpl whose id attribute is open (note the type attribute that makes it HTML)
  2. a JavaScript data object: here the argument of the success function: the JSON record provided by the server under route /json/*i*

```
{"Last": 'Twain',
 "First": 'Mark',
 "YoB": 1835,
  "YoD": 1910,
  "Title": 'Huckleberry␣Finn',
  "YoP": 1986,
  "Publisher": 'Penguin␣USA',
  "City": 'NY'}
```

JQuery Template Processing

- **Recall:** We are still trying to understand
  $("#content" + numb).loadTemplate($("#open''),data)
  It extends the empty <span id=''content*i*> in title.tpl with a details table:
- The loadTemplate method takes two arguments
  1. a template; here the result of $(#open),i.e. the element in bookshead.tpl whose id attribute is open (note the type attribute that makes it HTML)
  2. a JavaScript data object: here the argument of the success function: the JSON record provided by the server under route /json/*i*
- The JQuery template processing places the value of the data−content attribute into the <span>. The resulting table constitutes the generated "detail view":

```
<table>
  <tr>
    <th>Author:</th>
    <td>
      <span>Mark</span> <span>Twain</span>
      (<span>1835</span>−<span>1910</span>)
    </td>
  </tr>
  <tr>
    <th>Publisher:</th>
    <td><span>Penguin USA</span>, <span>NY</span></td>
  </tr>
</table>
```

JQuery Template Processing

- **Recall:** We are still trying to understand
  $("#content" + numb).loadTemplate($("#open''),data)
  It extends the empty <span id=''content*i*> in title.tpl with a details table:
- The loadTemplate method takes two arguments
  1. a template; here the result of $(#open),i.e. the element in bookshead.tpl whose id attribute is open (note the type attribute that makes it HTML)
  2. a JavaScript data object: here the argument of the success function: the JSON record provided by the server under route /json/*i*
- The JQuery template processing places the value of the data−content attribute into the <span>. The resulting table constitutes the generated "detail view":
- **Note:** Both the JavaScript object in step 2. as well as the result of the template processing show afterwards are virtual objects that exist only in memory. In particular, we do not have to write them explicitly.

Code: An AJAX-based Frontend for the Books App

- booksapp−ajax.py: the web server with two routes

```
import sqlite3
from bottle import route, run, template, static_file

# Connect to database
db = sqlite3.connect("./books.db")
# Row factory so we can have column names as keys.
db.row_factory = sqlite3.Row
cursor = db.cursor()

@route('/')
def books():
    cursor.execute('SELECT rowid, Title, YoP FROM Books')
    rv = cursor.fetchall()
    return template('titles', books=rv)

# JSON interfaces are very easy in bottle, just return a dictionary
@route('/json/<id:int>')
def book(id):
    cursor.execute(f'SELECT * FROM Books WHERE rowid={id}')
    row = cursor.fetchone() # Only one result, rowid is a primary key.
    return dict(zip(row.keys(), row)) # Pair up column names with values.

run(host='0.0.0.0', port=32500, debug=True)
```

10.4 Deploying the Books Application as a Program

Deploying The Books Application as a Program

- **Note:** Having a Python script booksapp.py you start with python3 booksapp.py is sufficient for development.
- If you want to deploy it on a web server, you want more: The sysadmin you deliver your web application to wants to start and manage it like any other `UNIX` command.
- **After all,** your web server will most likely be a `UNIX` (e.g. `linux`) computer.
- In particular behavioural variants should be available via command line options.
- **Example 4.1.** To run the books application without output (−q or −−quiet) and initialized with the seven book records we want to run

```
booksapp −q −−initbooks
```

Deploying The Books Application as a Program

- **Example 4.2.** If we forget the options, we need help:

```
> booksapp --help
Usage: <yourscript> [options]

Options:
  -h, --help show this help message and exit
  -q, --quiet don't␣print␣status␣messages␣to␣stdout
␣␣-l␣FILE,␣--log=FILE␣write␣log␣reports␣to␣FILE
␣␣--initbooks␣␣␣␣␣␣␣␣␣␣initialize␣with␣seven␣book␣records
```

Deploying a Python Script as a Shell Command/Executable

- We can make our a Python script behave like a native shell command.
- The file extension .py is only used by convention, we can leave it out and simply call the file booksapp.
- Then we can add a special Python comments in the first line

  ```
  #!/usr/bin/python3
  ```

  which the shell interprets as "call the program python3 on me".
- Finally, we make the file hello executable, i.e. tell the shell the file should behave like a shell command by issuing

  ```
  chmod u+x booksapp
  ```

  in the directory where the file booksapp is stored.
- We add the line

  ```
  export PATH="./:${PATH}"
  ```

  to the file .bashrc. This tells the shell where to look for programs (here the respective current directory called .)

Working with Options in Python

- We have the optparse library for dealing with command line options (install with pip3)
- **Example 4.3 (Options in the Books Application).**

```
from optparse import OptionParser
parser = OptionParser()
parser.add_option("-l", "--log", dest="logfile",
                  help="write logs to FILE", metavar="FILE")
parser.add_option("-q", "--quiet",
                  action="store_false", dest="verbose", default=True,
                  help="don't print status messages to stdout")
parser.add_option('--version',dest="version",default=1.0,type="float",
                  help="the version of the books application")

options, args = parser.parse_args()
# do something with the options and their args.
print ('VERSION   :', options.version)
```

Chapter 11
Image Processing

11.1 Basics of Image Processing

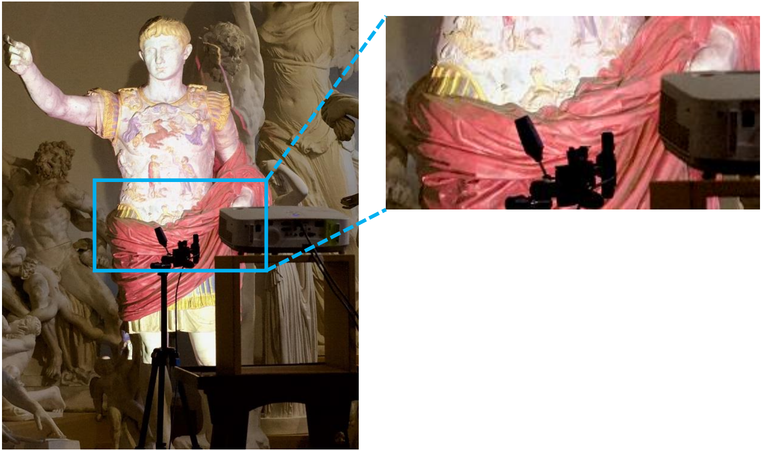
11.1.1 Image Representations

Images

- **Example 1.1 (Zooming in on Augustus).** A digital image taken by a standard DSLR camera. Let's zoom in on it!

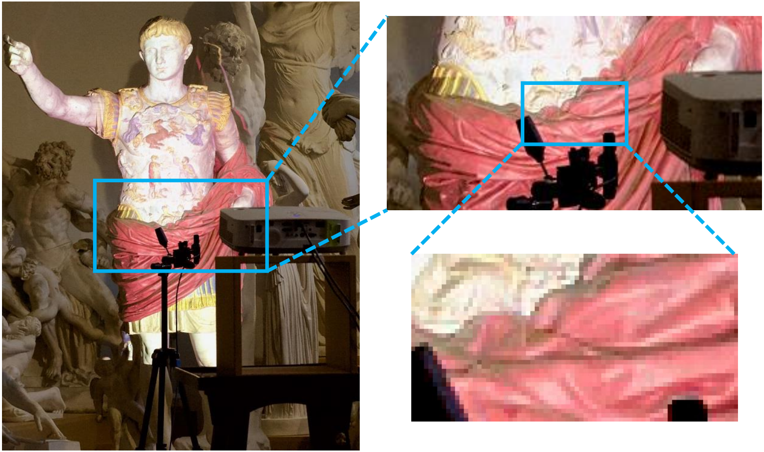
Images

- **Example 1.2 (Zooming in on Augustus).** And a bit more



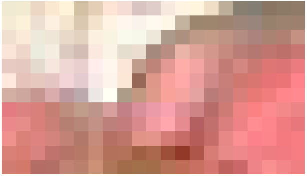
Images

- **Example 1.3 (Zooming in on Augustus).** When zooming in on an image, we start to see blocks of colors, which are organized in a regular grid.



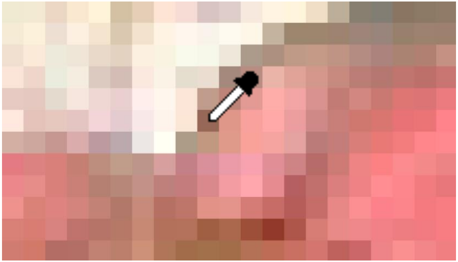
Images as Rasters of Pixels

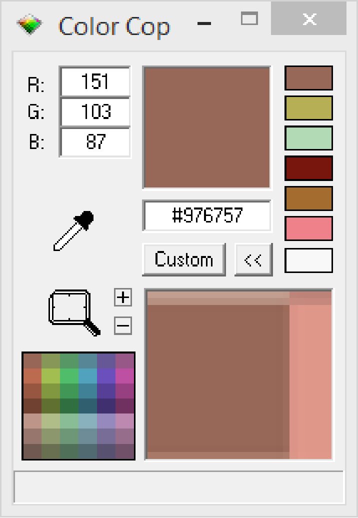
- If we zoom in quite a bit more, we see
- **Observation:** The colors are arranged in a two- dimensional grid (raster).



- **Definition 1.4.** We call the grid raster and each entry in it pixel (from "picture element").

Colors





- **Definition 1.5.** Colors are usually represented in RGB format, i.e. as triples $\langle R, G, B\rangle$ with three channels (also called bands).
- $R, G, B \in [0{,}255] \leadsto$ One Byte per channel per pixel.
- Images in this format can store $256 \cdot 256 \cdot 256 = 256^3$ (about 16 million) colors.

Color Examples

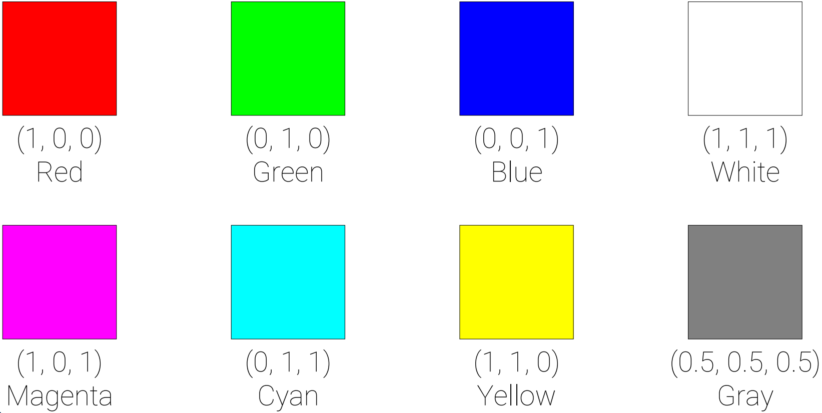
- **Example 1.6.** A color can be represented by three numbers.

(255, 0, 0) Red	(0, 255, 0) Green	(0, 0, 255) Blue	(255, 255, 255) White
(255, 0, 255) Magenta	(0, 255, 255) Cyan	(255, 255, 0) Yellow	(128, 128, 128) Gray

- **Definition 1.7.** A color is called grayscale, iff $R = G = B$

Normalized Color Values

- **Observation 1.8.** *For color representations, only the relative contribution of the band is imporant.*
- **Definition 1.9.** Normalized colors use pixel values between 0 and 1.
- **Idea:** Values are still stored as Bytes, but normalized before use: $v' = v/255$
- **Example 1.10.**



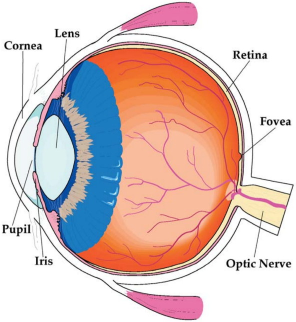
HTML Color Codes

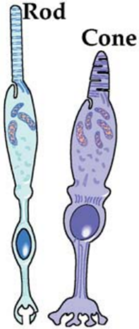
- HTML uses a shorthand notation for colors using hexadecimal numbers.
- **Example 1.11.**

#FF0000	#00FF00	#0000FF	#FFFFFF
Red	Green	Blue	White
#FF00FF	#00FFFF	#FFFF00	#808080
Magenta	Cyan	Yellow	Gray

The Human Eye

- **Definition 1.12 (The Human Eye).** Light from our surroundings enters our eye through the lens and then hits the retina on the back of our eye.



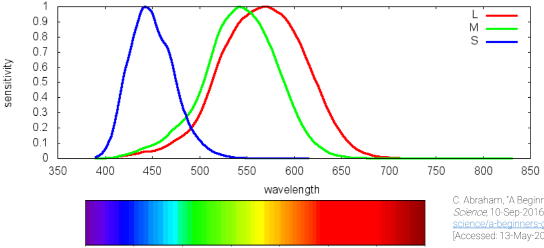


The retina has cones and rods, which are responsible for color and brightness vision, respectively.

- Since we are interested in colors here, we will ignore the rods for the purpose of this lecture.

The Human Eye – Three Types of Cones

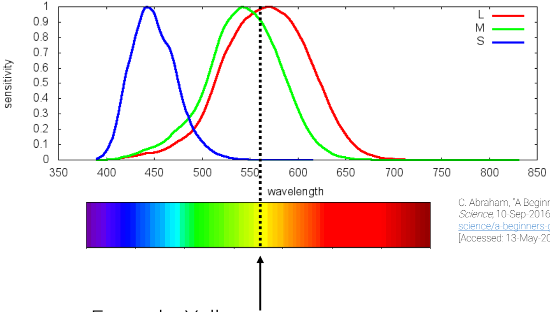
- **Sensitivity of the Three Cones:**



C. Abraham, "A Beginner's Guide to (CIE) Colorimetry," *Hipster Color Science*, 10-Sep-2016. Available: https://medium.com/hipster-color-science/a-beginners-guide-to-colorimetry-401f1830b65a. [Accessed: 13-May-2019].

The Human Eye – Three Types of Cones

- **Example 1.13 (We see Yellow).**



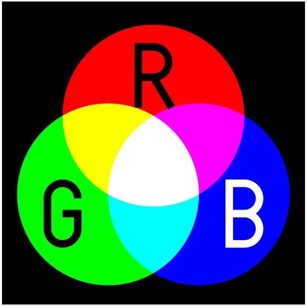
C. Abraham, "A Beginner's Guide to (CIE) Colorimetry," *Hipster Color Science*, 10-Sep-2016. Available: https://medium.com/hipster-color-science/a-beginners-guide-to-colorimetry-401f1830b65a. [Accessed: 13-May-2019].

Example: Yellow
Both "red" and "green" cone are stimulated.

- **Observation 1.14.** *We can create all (human-visible) colors as a mixture of red, green, and blue light.*

Monitors

- **Definition 1.15.** A computer monitor (or just monitor)is an output device for visual information.
- Monitors (usually) have pixels, too!
- **Definition 1.16.** In color monitors, pixels typically consist not of a single light source, but three distinct subpixels.
- If these subpixels are small enough and close together, our eye cannot see that the light actually comes from different points and thus perceives the mixture color.



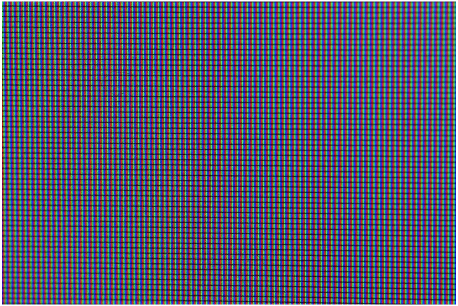


Image Size

- **Example 1.17 (Augustus again).**

  Image: 1440×746 pixels
  Expected file size:
  $\mathrm{Width} \cdot \mathrm{Height} \cdot \mathit{Channels}$
  $1440 \cdot 746 \cdot 3 = 3,222,720\mathrm{B} \approxeq 3\mathrm{MiB}$

- But if we look onto our disk we see somthing completely different:

Augustus.jpg	4/30/2019 2:58 PM	JPEG image	404 KB
Augustus.png	6/3/2019 12:19 PM	PNG image	1,628 KB

- On disk, images are usually compressed (JPEG, PNG, GIF,WebP etc). JPEG file size is smaller than PNG, but image quality is lost.

JPEG Compression Artefacts

- **Example 1.18 (Augustus again).** Here, the Augustus image is saved with a very high jpeg compression. The file size is tiny (27 KB, compare to 440 KB on previous slide). However, the image quality suffers.
  JPEG creates blocks of pixels, and approximates the colors in this block with as few bits as possible (according to compression ratio).

AugustusCompressed.jpg 6/7/2019 9:11 AM JPEG image 27 KB

11.1.2 Basic Image Processing in Python

The Pillow Library for Image Processing in Python

- We will use the Pillow library in IWGS.
- **Definition 1.19.** Pillow is a fork (a version) of the old Python library PIL (Python Image Library). (hence the name)
- Details at `https://pillow.readthedocs.io/slides/stable/`
- **Install:** pip install Pillow
- **Example 1.20.** Determine the color of a particular pixel

```
from PIL import Image
# load image
im = Image.open('image.jpg')
im.show()
# access color at pixel (x, y)
x = 15
y = 300
r, g, b = im.getpixel((x, y))
```

The Pillow Library for Image Processing in Python

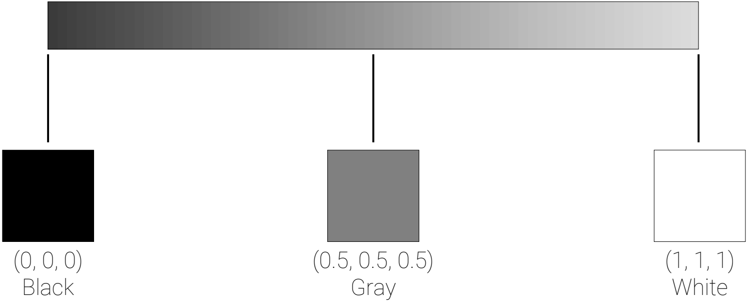
- We will use the Pillow library in IWGS.
- **Definition 1.22.** Pillow is a fork (a version) of the old Python library PIL (Python Image Library). (hence the name)
- Details at `https://pillow.readthedocs.io/slides/stable/`
- **Install:** pip install Pillow
- **Example 1.24.** Directly use the image object in jupyter notebooks:

```
from PIL import Image
# load image
im = Image.open('image.jpg')
im # in Jupyter Notebooks, we can directly use the variable
```

The notebooks shows the image in a new cell.

Grayscale Images

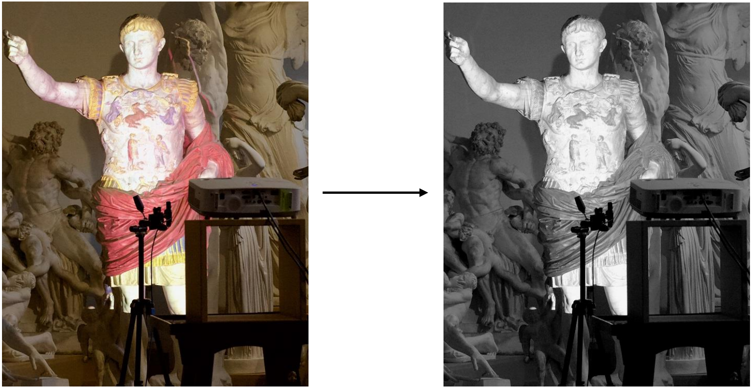
- **Recall:** A color is grayscale, iff R=G=B.



- **Idea:** If all channels have the same value, why store all three?
- Grayscale images usually have only one channel.

Grayscale Conversion

- **Observation 1.25.** *Humans are very sensitive to green, less to red, and least to blue.*
- **Definition 1.26.** To convert an image to an grayscale image (grayscale conversion), we compute $Gray = 0.21R + 0.71G + 0.08B$
- **Example 1.27 (Grayscale Conversion).**



More Image Operations

- **Example 1.28 (More Image Operations).**


Original

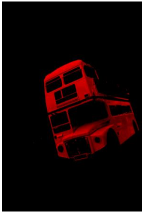

Grayscale


Sepia


Inverse

Each pixel is
processed separately!


Threshold


Red Channel
Extraction

- As for grayscale conversion of these process each pixel separately.

Image Operations in Pillow

- The pillow library supports many image operations out of the box.
- **Example 1.29 (Grayscale Conversion and Inversion in Pillow).**

```
from PIL import Image, ImageOps
im = Image.open ('image.jpg')
# convert to grayscale
gray = ImageOps.grayscale(im)
# invert image
inverse = ImageOps.invert(im)
```

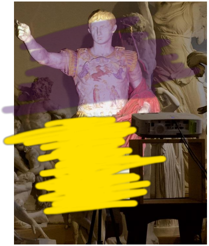
- Complete List: `https://pillow.readthedocs.io/en/stable/reference/ImageOps.html`

Transparency and Image Composition

- Sometimes we want to overlay images ⇝ layers.
- We need a notion of how transparent a pixel is.
- **Definition 1.30.** We introduce a fourth channel: A (for alpha). Alpha is the opacity (inverse of transparency). A pixel is now $\langle R, G, B, A\rangle$.
- **Example 1.31 (Combining Images).**

+

=



- **Note:** The order of layers is important here: The Augustus image is below the other image! The Augustus image has *no* transparency, the second image does!

Transparency (continued)

- **Example 1.32 (Combining Images).**

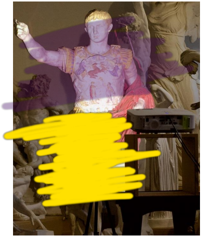
(R,G,B,A) = (0.6, 0.0, 1.0, 0.5)
Half transparent purple

(R,G,B,A) = (0, 0, 0, 0)
Full transparent

(R,G,B,A) = (1, 1, 0, 1)
Full yellow

+

=



$$R_{target} = (1-A) \times R_{augustus} + A \times R_{purple,yellow}$$

$$G_{target} = (1-A) \times G_{augustus} + A \times G_{purple,yellow}$$

$$B_{target} = (1-A) \times B_{augustus} + A \times B_{purple,yellow}$$

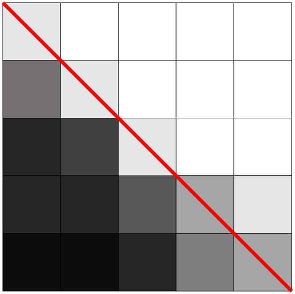
11.1.3 Edge Detection

Edge Detection

- **Goal:** Find interesting parts of image (features).

Edge Detection

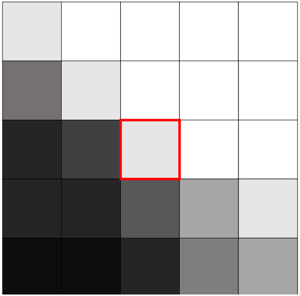
- **Goal:** Find interesting parts of image (features).
- **Example 1.35 (Edge Detection).** Find edges, i.e. image sections, where color changes rapidly.



Clearly there is an edge in this image. How do we detect it automatically?

Edge Detection

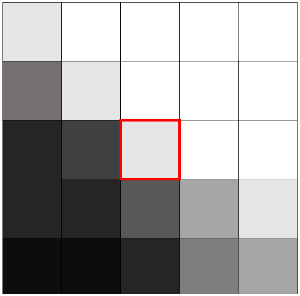
- **Goal:** Find interesting parts of image (features).
- **Example 1.37 (Edge Detection).** Find edges, i.e. image sections, where color changes rapidly.



Decide for each pixel, whether it is on an edge. Here: Is marked pixel an edge pixel?

Edge Detection

- **Goal:** Find interesting parts of image (features).
- **Example 1.39 (Edge Detection).** Find edges, i.e. image sections, where color changes rapidly.



  Inspect neighbor pixels.

Edge Detection

- **Goal:** Find interesting parts of image (features).
- **Example 1.41 (Edge Detection).** Find edges, i.e. image sections, where color changes rapidly.
- **Definition 1.42.** We call a pixel a horizontal edge pixel, iff

$$I_B - I_T + I_{BL} - I_{TL} + I_{BR} - I_{TR} > \tau$$

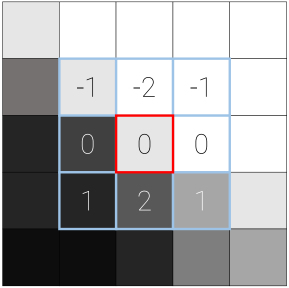
for some threshold τ and a vertical edge pixel, iff

$$I_R - I_L + I_{TR} - I_{TL} + I_{BR} - I_{BL} > \tau$$

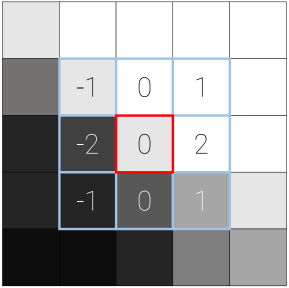
Algorithm: Sobel Filter

- **Idea:** There is a general algorithm that computes this.
- **Definition 1.43.** Given a 3×3 matrix M, the Sobel filter computes a new pixel value by getting the pixel value of each neighbor in 3x3 window, multiply with the components in M and adding everything up.
- **Observation 1.44.** *Given a suitable matrix M, the Sobel filter computes the quantities from 1.34.*
- **Example 1.45 (Edge Tests via Sobel Filters).**

Horizontal edge test:



Vertical edge test:

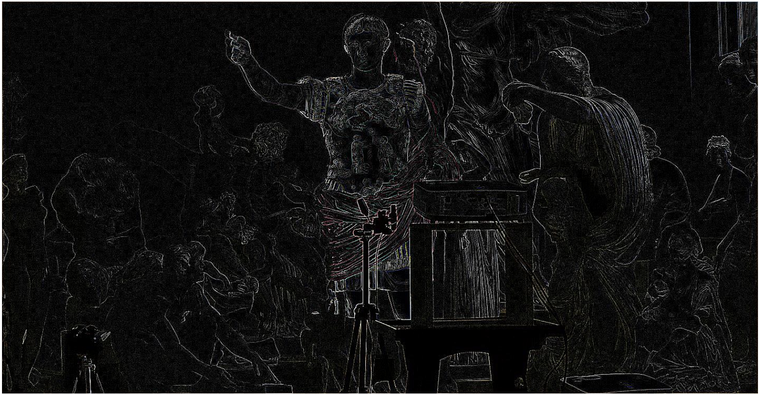


Edge-Detection in Pillow

- **Example 1.46 (Augustus and his Edges).**

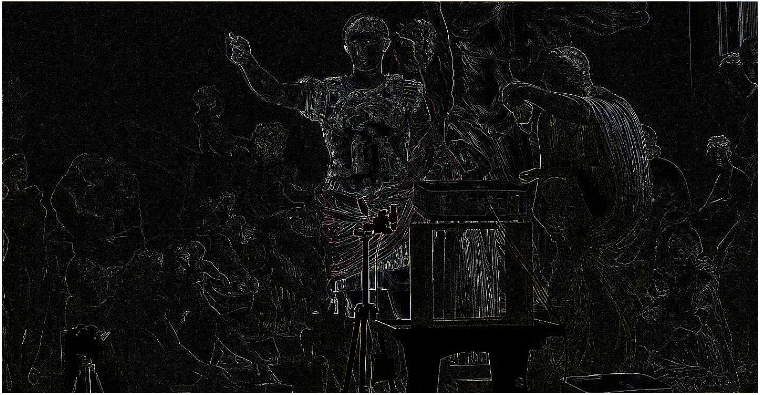
Edge-Detection in Pillow

- **Example 1.48 (Augustus and his Edges).**



Edge-Detection in Pillow

- **Example 1.50 (Augustus and his Edges).**



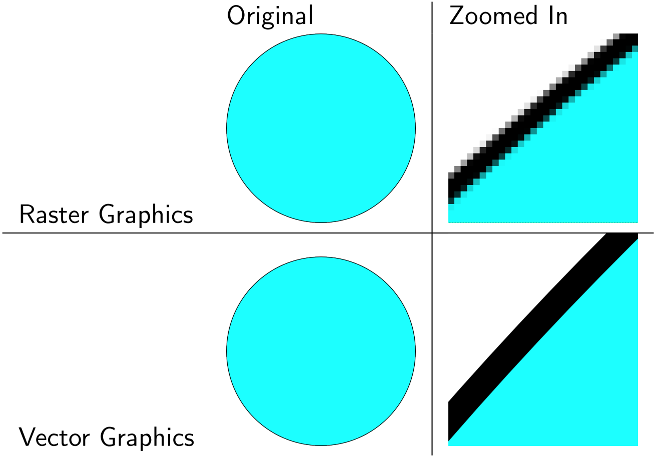
- **Example 1.51 (Edge Detection in Pillow).**

```
from PIL import Image, ImageFilter
im = Image.open('augustus.jpg')
edges = im.filter(ImageFilter.FIND_EDGES)
edges.show() # or just edges in Jupyter
```

11.1.4 Scalable Vector Graphics

Vector Graphics

- **Problem:** Raster images store colors in pixel grid. Quality deteriorates when image is zoomed into.
- Vector Graphics solve this problem!



Vector Graphics (Definition)

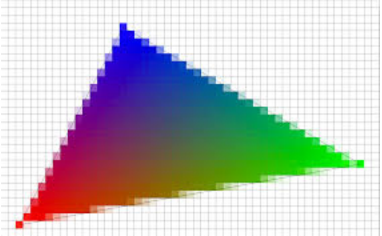
- **Definition 1.52.** Image representation formats that store shape information instead of individual pixels, are refered to as vector graphics.
- **Example 1.53.** For a circle, just store
  - center
  - radius
  - line width
  - line color
  - fill color
- **Example 1.54.** For a line, store
  - start and end point
  - line width
  - line color

Vector Graphics Display

- There are devices that directly display vector graphics.
- **Example 1.55.**

Vector Graphics Display

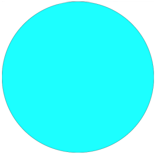
- There are devices that directly display vector graphics.
- **Example 1.58.**
- **Definition 1.59.** For monitors, vector graphics must be rasterized – i.e. converted into a raster image before display.
- **Example 1.60.**



Scalable Vector Graphics (SVG)

- **Definition 1.61.** Scalable Vector Graphics (SVG) is an XML-based markup format for vector graphics.
- **Example 1.62.**

```
<svg xmlns="http://www.w3.org/2000/svg"
     width="100" height="100" >
  <circle cx="50" cy="50" r="50"
      style="fill:#1cffff;␣stroke:#000000;␣stroke−width:0.1" />
</svg>
```



  - The <svg> tag starts the SVG document, width, height declare its size.
  - The <circle> tag starts a circle. cx, cy is the center point, r is the radius. style describes how the circle looks.

  As the SVG size is 100x100 and the circle is at (50,50) with radius 50, it is centered and fills the whole region.

More SVG Primitives

- **Example 1.63 (Rectangle).**

  ```
  <rect x="..." y="..." width="..." height="..." style="..." />
  ```

- **Example 1.64 (Ellipse).**

  ```
  <ellipse cx="..." cy="..." rx="..." ry="..." style="..." />
  ```

- **Example 1.65 (Line).**

  ```
  <line x1="..." y1="..." x2="..." y2="..." style="..." />
  ```

- **Example 1.66 (Text).**

  ```
  <text x="..." y="..." style="...">This is my text!</text>
  ```

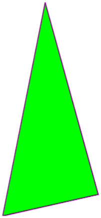
- **Example 1.67 (Image).**

  ```
  <image xlink:href="..." x="..." y="..." width="..." height="..." />
  ```

SVG Polygons

- **Example 1.68 (An SVG Triangle).**

```
<svg height="210" width="500" xmlns="http://www.w3.org/2000/svg">
  <polygon points="200,10 250,190 160,210"
           style="fill:lime;stroke:purple;stroke−width:1"/>
</svg>
```



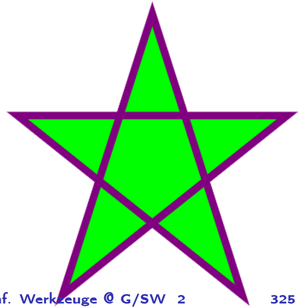
SVG Polygons

- **Example 1.70 (An SVG Triangle).**

```
<svg height="210" width="500" xmlns="http://www.w3.org/2000/svg">
  <polygon points="200,10 250,190 160,210"
           style="fill:lime;stroke:purple;stroke-width:1"/>
</svg>
```

- **Example 1.71 (An SVG Pentagram).**

```
<svg height="210" width="210" xmlns="http://www.w3.org/2000/svg">
  <polygon points="100,10 40,198 190,78 10,78 160,198"
           style="fill:lime;stroke:purple;stroke-width:5;fill-rule:nonzero;"/>
</svg>
```



SVG in HTML

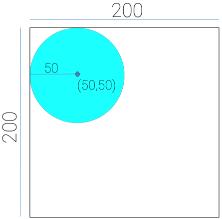
- SVG can be used in dedicated files (file ending .svg) and referenced in a <img> tag.
- It can however also be written directly in HTML files.
- **Example 1.72.** Triangle from 1.68 embedded in HTML file

```
<html>
  <body>
    <svg height="210" width="500" xmlns="http://www.w3.org/2000/svg">
      <polygon points="200,10␣250,190␣160,210"
          style="fill:lime;stroke:purple;stroke−width:1" />
    </svg>
  </body>
</html>
```

The SVG viewBox Attribute

- **Idea:** The SVG viewBox attribute allows us to zoom into an image.
- **Example 1.73.**

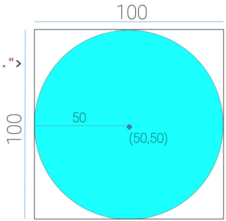
```
<svg width="200" height="200" xmlns="...">
  <circle cx="50" cy="50" r="50" style="..." />
</svg>
```



Here, the width and height are scaled by a factor of 2 to give us a little more room. Sometimes we want to specify a larger image, but only display a section of it.

- **Example 1.74.**

```
<svg width="200" height="200" xmlns="..."
     viewBox="0␣0␣100␣100" >
  <circle cx="50" cy="50" r="50" style="..." />
</svg>
```



viewBox specifies a region inside our canvas. Only things inside that are drawn. The resulting image is then stretched to the canvas size (zoom effect).

11.2 Project: An Image Annotation Tool

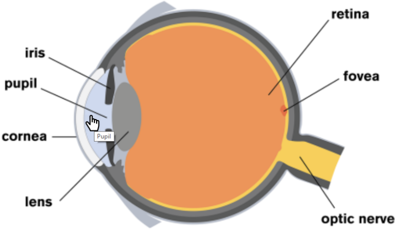
Project: Kirmes Image Annotation Tool

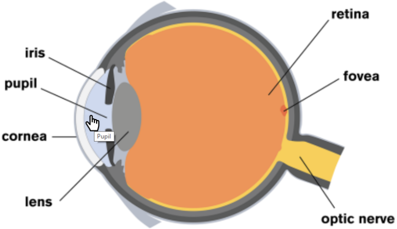
- **Problem:** Our Books-App project was a fully functional web application, but does not do anything useful for DigiHumS.
- **Idea:** Extend/Adapt it to a database for image annotation like LabelMe [LM].
- **Setting:** Prof. Peter Bell (formerly at FAU) conducts research on baroque paintings on parish fairs (Kirmes) and the iconography in these paintings. We want to build an annotation system for this research.
- **Project Goals:**
  1. Collect kirmes images in a database and display them,
  2. mark interesting areas and provide meta data,
  3. display/edit/search annotated information.

  1. is analogous to Books-App, for 2/3. we need to know more
- **Plan:** Lern the necessary technologies in class, build the system in exercises

HTML Image Maps

- **Definition 2.1.** HTML image maps mark areas in an digital image and assign names and links to them.
- **Example 2.2.** An image map adds hover and on click behavior





Clicking on the pupil leads to:
`https://en.wikipedia.org/wiki/Pupil`

Clicking on the vitreous body leads to:
`https://en.wikipedia.org/wiki/Vitreous_body`

HTML Image Maps

- **Definition 2.3.** HTML image maps mark areas in an digital image and assign names and links to them.
  - **Example 2.4.** An image map adds hover and on click behavior

```html
<html>
  <body>
    <img src="Human_Eye_Structures.png" usemap="#image-map"/>
    <map name="image-map">
      <area title="Pupil"
            href="https://en.wikipedia.org/wiki/Pupil"
            coords="102,117,143,219" shape="rect"/>
      <area title="Vitreous␣Body"
            href="https://en.wikipedia.org/wiki/Vitreous_body"
            coords="242,166,107" shape="circle"/>
    </map>
  </body>
</html>
```

- Easy creation of image maps: `https://www.image-map.net/`

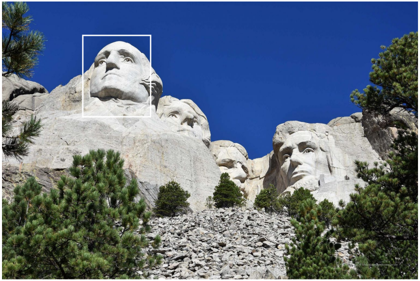
Problems of HTML Image Maps

- **Problem:** Image maps do not allow interaction:
  - the name attribute can only contain unstructured information.
  - no integrated highlight for image maps area,
  - no onclick or onmouseover attributes.

  But the whole point is to have (arbitrarily) complex metadata for image regions.
- **New Plan:** Use a newer technology: SVG and CSS.

Handcrafting better Image Annotations with SVG and CSS

- **Idea:** Integrate the image and the areas into one SVG and make areas interactive via CSS.
- **Example 2.5 (Paper Prototype).** Highlight regions and display information on hover.



George Washington

Abraham Lincoln

SVG Annotation Implementation Areas

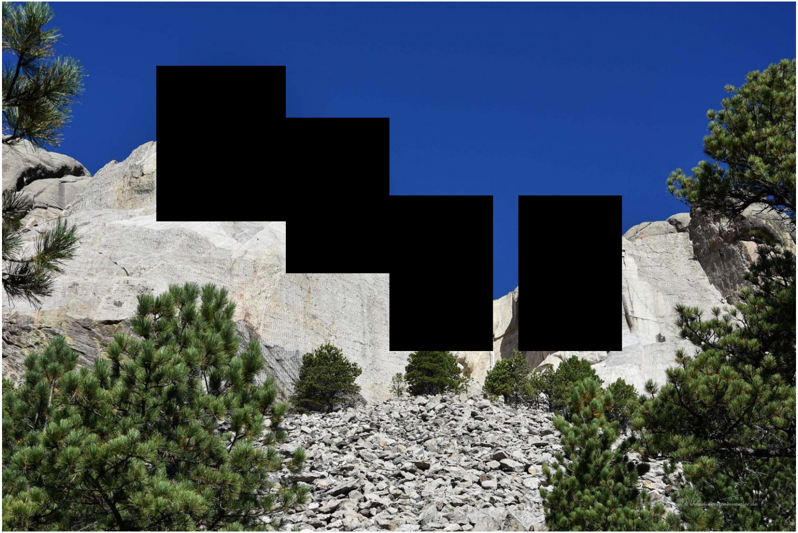
- **Implementing Areas as Rectangles:**

```
<svg xmlns="http://www.w3.org/2000/svg" width="1536" height="1024" >
  <!-- Image -->
  <image width="1536" height="1024" xlink:href="mount_rushmore.jpg" />
  <!-- Areas in image as rects. -->
  <rect x="300" y="125" width="250" height="300"/>
  <rect x="550" y="225" width="200" height="300"/>
  <rect x="750" y="375" width="200" height="300"/>
  <rect x="999" y="375" width="200"height="300"/>
</svg>
```

Add four <rect>s (one for each president).

SVG Annotation Implementation Result

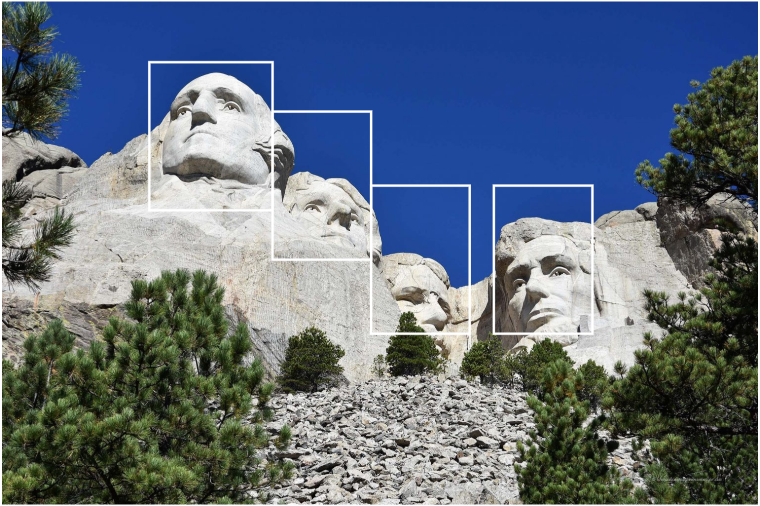
- **Areas as Rectangles – Result:** Now the rectangles are visible



Adding CSS for the Areas

- **Example 2.6 (Adding CSS).**

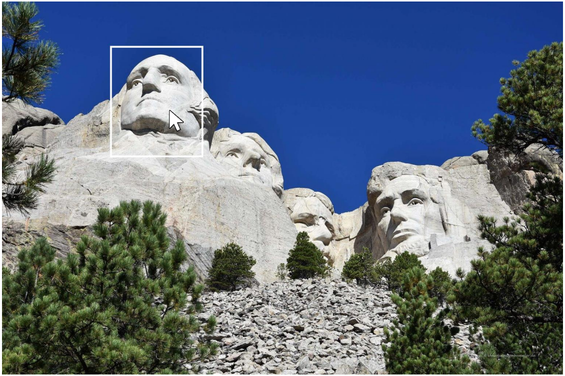
  ```
  rect {fill—opacity:0; stroke:white; stroke—opacity:1; stroke—width:5px}
  ```



Selectively Highlighting Areas

- **Problem:** Now the rectangles are always visible.
- **Idea:** make the rectangles invisible by default only show them on hover.
- **CSS:** We set the stroke opacity to zero by default and add a hover selector.

```
rect {fill–opacity:0; stroke:white; stroke–opacity:0; stroke–width:5px}
rect:hover {stroke–opacity:1}
```



Adding Annotation Text

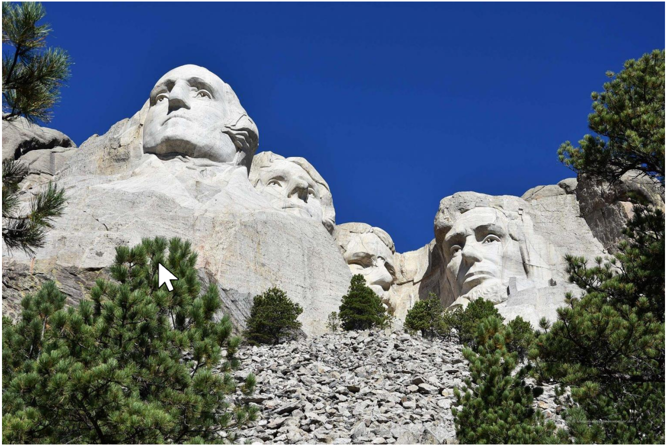
- **Adding Annotation Text** and making space for it.

```
<svg xmlns="http://www.w3.org/2000/svg" width="1536" height="1224" >
  <!-- Image -->
  <image width="1536" height="1024" xlink:href="mount_rushmore.jpg" />
  <!-- Areas in image as rects, text below -->
  <rect x="300" y="125" width="250" height="300" />
  <text x="100" y="1200">George Washington</text>
  <rect x="550" y="225" width="200" height="300" />
  <text x="100" y="1200">Thomas Jefferson</text>
  <rect x="750" y="375" width="200" height="300" />
  <text x="100" y="1200">Theodore Roosevelt</text>
  <rect x="999" y="375" width="200" height="300" />
  <text x="100" y="1200">Abraham Lincoln</text>
</svg>
```

and we add some CSS:

```
text {fill:black; opacity:1; font-size:100px}
```

- **Adding Annotation Text – Result:**



Selectively Showing Annotations

- **Problem:** Now the annotations are always visible.
- **Idea:** Add CSS hover effect for <rect>s, which effects the |<text>|.
- **Definition 2.7.** The CSS sibling operator + modifies a selector so that it (only) affects following sibling elements (same level).
- **Example 2.8.** In the CSS directive

  ```
  rect:hover + text {<rules>}
  ```

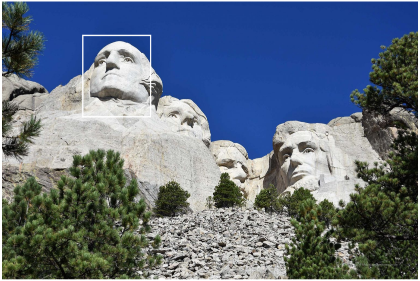
  Selector Sibling operator Target

  the rules affect the SVG <text> directly after the <rect> element.
- **Again:** the order of elements in the HTML is important!
- **CSS:** We set the opacity to zero by default and add a hover selector for the following <text> sibling.

  ```
  text {fill:black; opacity:0; font-size:100px}
  rect:hover + text {opacity: 1}
  ```

Image Annotation Tool – Final Result

- Now our annotation tool works as expected!
- **Example 2.9 (Final Result).** Highlight regions and display information on hover.



George Washington

Abraham Lincoln

11.3 Fun with Image Operations: CSS Filters

CSS Image Filters

- **Goal:** Apply image filters (grayscale etc.) directly in CSS.
- **Example 3.1 (Image Effects via inline CSS).**

```
<img style="filter: grayscale(100%)" src="augustus.jpg" alt="no image"/>
```

- **Disadvantage:** The original image is delivered to client. When user saves the image, they get the original!

Some more CSS Filters

- **Example 3.2 (Image Effects via CSS Style sheets).**

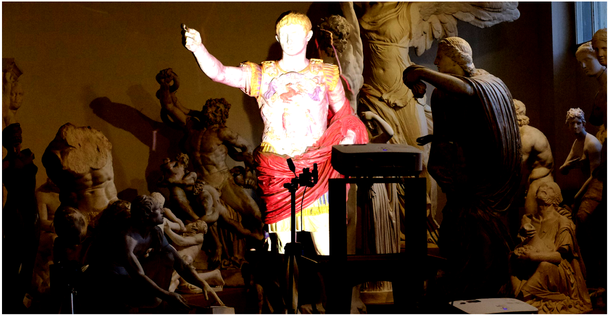
  ```
  <img style="filter:␣blur(4px" src="augustus.jpg" alt="no␣image"/>
  ```

Some more CSS Filters

- **Example 3.3 (Image Effects via CSS Style sheets).**

```
<img style="filter:␣blur(4px" src="augustus.jpg" alt="no␣image"/>
```

```
<img style="filter:␣contrast(180%)" src="augustus.jpg" alt="no␣image"/>
```



Some more CSS Filters

- **Example 3.4 (Image Effects via CSS Style sheets).**

```
<img style="filter: blur(4px" src="augustus.jpg" alt="no image"/>
```

```
<img style="filter: contrast(180%)" src="augustus.jpg" alt="no image"/>
```

```
<img style="filter: hue-rotate(90deg)" src="augustus.jpg" alt="no image"/>
```

Combining CSS Filters

- **Idea:** We can also combine image filters flexibly. The easist way is when we define CSS classes for that.
- **Example 3.5 (Tie CSS Filters to Classes).**

```
<html>
  <head>
    <style type="text/css">
      .blur { filter: blur(4px); }
      .brightness { filter: brightness(0.30); }
      .contrast { filter: contrast(180%); }
      .grayscale { filter: grayscale(100%); }
      .huerotate { filter: hue-rotate(180deg); }
      .invert { filter: invert(100%); }
      .opacity { filter: opacity(50%); }
      .saturate { filter: saturate(7); }
      .sepia { filter: sepia(100%); }
      .shadow { filter: drop-shadow(8px 8px 10px green); }
    </style>
  </head>
  <body>
    <img class="blur invert saturate" alt="no image" src="augustus.jpg" />
  </body>
</html>
```

Filtering Everyghing Else

- **Note:** CSS filters don't just apply to images! (Almost) everything can be filtered.
- **Example 3.6 (Filtering Text (Blurring)).**

```
<p style="filter:␣blur(3px)">A severely blurred Text</p>
```

CSS Animations

- **Definition 3.7.** CSS animations change state of an object over time.
- **Example 3.8 (Inverting an image).**

```
img {animation: invertAnimation 1s forwards}

@keyframes invertAnimation {
    from {filter: none}
    to {filter: invert(100%)}
  }
```

SVG Filters

- **Note:** Unfortunately in SVG the filtering works differently from CSS.
- **Example 3.9 (Blurring Mt. Rushmore in SVG).**

```
<svg xmlns="http://www.w3.org/2000/svg" width="1536" height="1024">
  <style> image {filter: url(#myCustomFilter)}</style>
  <image width="1536" height="1024" xlink:href="mount_rushmore.jpg" />
  <!-- Image filter -->
  <filter id="myCustomFilter">
    <feGaussianBlur stdDeviation="5" />
  </filter>
</svg>
```

- **Example 3.10 (SVG Filters can be combined).**

```
  <filter id="myCustomFilter">
    <feGaussianBlur stdDeviation="5" />
    <feColorMatrix type="saturate" values="0.1" />
  </filter>
```

Chapter 12
Ontologies, Semantic Web for Cultural Heritage

12.1 Documenting our Cultural Heritage

Documenting our Cultural Heritage

- **Definition 1.1.** Cultural heritage is the legacy of physical artifacts cultural artefacts and practices, representations, expressions, knowledge, or skills – intangible cultural heritage (ICH) of a group or society that is inherited from past generations.
- **Problem:** How can we understand, conserve, and learn from our cultural heritage?
- **Traditional Answer:** We collect cultural artefacts, study them carefully, relate them to other artefacts, discuss the findings, and publish the results. We display the artefacts in museums and galleries, and educate the next generation.
- **DigHumS Answer:** In "Digital Humanities and Social Sciences", we want to represent our cultural heritage digitally, and utilize computational tools to do so.
- **Practical Question:** What are the best representation formats and tools?

Research Data in a Nutshell

- **Definition 1.2.** Research data is any information that has been collected, observed, generated or created to validate original research findings. Although usually digital, research data also includes non-digital formats such as laboratory notebooks and diaries.
- **Types of research data:**
  - documents, spreadsheets, laboratory notebooks, field notebooks, diaries,
  - questionnaires, transcripts, codebooks, test responses,
  - audiotapes, videotapes, photographs, films,
  - cultural artefacts, specimens, samples,
  - data files, database contents (video, audio, text, images), digital outputs,
  - models, algorithms, scripts,
  - contents of an application (input, output, logfiles, schemata),
  - methodologies and workflows, standard operating procedures, and protocols,
- **Non-digital Research Data** such as cultural artefacts, laboratory notebooks, ice-core samples, or sketchbooks is often unique. Materials could be digitized, but this may not be possible for all types of data.

FAIR Research Data: The Next Big Thing

- **Principle:** Scientific experiments must be replicated, and derivations must be checkable to be trustworthy. (consensus of scientific community)
- **Intuition:** Research data must be retained for justification, shared for synergies!
- **Consequence:** Virtually all scientific funding agencies now require some kind of research data strategy in proposals. (tendency: getting stricter)

FAIR Research Data: The Next Big Thing

- **Principle:** Scientific experiments must be replicated, and derivations must be checkable to be trustworthy. (consensus of scientific community)
- **Intuition:** Research data must be retained for justification, shared for synergies!
- **Consequence:** Virtually all scientific funding agencies now require some kind of research data strategy in proposals. (tendency: getting stricter)
- **Problem:** Not all forms of data are actually useable in practice.
- **Definition 1.4 (Gold Standard Criteria).** Research data should be FAIR:
  - Findable: easy to identify and find for both humans and computers, e.g. with metadata that facilitate searching for specific datasets,
  - Accessible: stored for long term so that they can easily be accessed and/or downloaded with well-defined access conditions, whether at the level of metadata, or at the level of the actual data,
  - Interoperable: ready to be combined with other datasets by humans or computers, without ambiguities in the meanings of terms and values,
  - Reusable: ready to be used for future research and to be further processed using computational methods.

  Consensus in the research data community; for details see [FAIR18; Wil+16].

FAIR Research Data: The Next Big Thing

- **Principle:** Scientific experiments must be replicated, and derivations must be checkable to be trustworthy. (consensus of scientific community)
- **Intuition:** Research data must be retained for justification, shared for synergies!
- **Consequence:** Virtually all scientific funding agencies now require some kind of research data strategy in proposals. (tendency: getting stricter)
- **Problem:** Not all forms of data are actually useable in practice.
- **Definition 1.5 (Gold Standard Criteria).** Research data should be FAIR:
  - Findable: easy to identify and find for both humans and computers, e.g. with metadata that facilitate searching for specific datasets,
  - Accessible: stored for long term so that they can easily be accessed and/or downloaded with well-defined access conditions, whether at the level of metadata, or at the level of the actual data,
  - Interoperable: ready to be combined with other datasets by humans or computers, without ambiguities in the meanings of terms and values,
  - Reusable: ready to be used for future research and to be further processed using computational methods.

  Consensus in the research data community; for details see [FAIR18; Wil+16].
- **Open Question:** How can we achieve FAIR-ness in a discipline in practice?

Categories of Data in DigiHumS and their Formats

- We distinguish four broad categories of data in DigiHumS.
- **Definition 1.6.** Concrete data: digital representations of artefacts in terms of simple data,
  - e.g. raster images as pixel arrays in JPEG. (see)
  - e.g. books identified by author/title/publisher/pubyear. (see)

Categories of Data in DigiHumS and their Formats

- We distinguish four broad categories of data in DigiHumS.
- **Definition 1.12.** Concrete data: digital representations of artefacts in terms of simple data,
  - e.g. raster images as pixel arrays in JPEG. (see)
  - e.g. books identified by author/title/publisher/pubyear. (see)
- **Definition 1.13.** Narrative data: documents and text fragments used for communicating knowledge to humans.
  - e.g. plain text and formatted text with markup code (see 4 (Documents as Digital Objects) in the IWGS lecture notes)

Categories of Data in DigiHumS and their Formats

- We distinguish four broad categories of data in DigiHumS.
- **Definition 1.18.** Concrete data: digital representations of artefacts in terms of simple data,
  - e.g. raster images as pixel arrays in JPEG. (see)
  - e.g. books identified by author/title/publisher/pubyear. (see)
- **Definition 1.19.** Narrative data: documents and text fragments used for communicating knowledge to humans.
  - e.g. plain text and formatted text with markup code (see 4 (Documents as Digital Objects) in the IWGS lecture notes)
- **Definition 1.20.** Symbolic data: descriptions of object and facts in a formal language
  - e.g. 3+5 in Python (see 2 (Introduction to Programming) in the IWGS lecture notes)

Categories of Data in DigiHumS and their Formats

- We distinguish four broad categories of data in DigiHumS.
- **Definition 1.24.** Concrete data: digital representations of artefacts in terms of simple data,
  - e.g. raster images as pixel arrays in JPEG. (see)
  - e.g. books identified by author/title/publisher/pubyear. (see)
- **Definition 1.25.** Narrative data: documents and text fragments used for communicating knowledge to humans.
  - e.g. plain text and formatted text with markup code (see 4 (Documents as Digital Objects) in the IWGS lecture notes)
- **Definition 1.26.** Symbolic data: descriptions of object and facts in a formal language
  - e.g. 3+5 in Python (see 2 (Introduction to Programming) in the IWGS lecture notes)
- **Definition 1.27.** Metadata: "data about data", e.g. who has created these facts, images, or documents, how do they relate to each other?(not covered yet)
- **Observation 1.28.** *Metadata are the resources, DigiHumS results are made of ($\rightsquigarrow$ support that)*
  The other categories digitize artefacts and auxiliary data.

Categories of Data in DigiHumS and their Formats

- We distinguish four broad categories of data in DigiHumS.
- **Definition 1.30.** Concrete data: digital representations of artefacts in terms of simple data,
  - e.g. raster images as pixel arrays in JPEG. (see)
  - e.g. books identified by author/title/publisher/pubyear. (see)
- **Definition 1.31.** Narrative data: documents and text fragments used for communicating knowledge to humans.
  - e.g. plain text and formatted text with markup code (see 4 (Documents as Digital Objects) in the IWGS lecture notes)
- **Definition 1.32.** Symbolic data: descriptions of object and facts in a formal language
  - e.g. 3+5 in Python (see 2 (Introduction to Programming) in the IWGS lecture notes)
- **Definition 1.33.** Metadata: "data about data", e.g. who has created these facts, images, or documents, how do they relate to each other?(not covered yet)
- **Observation 1.34.** *Metadata are the resources, DigiHumS results are made of (⇝ support that)*
  The other categories digitize artefacts and auxiliary data.
- **Observation 1.35.** *We will need all of these – and their combinations – to do DigiHumS.*

WissKI: a Virtual Research Env. for Cultural Heritage

- **Definition 1.36.** WissKI is a virtual research environment (VRE) for managing scholarly data and documenting cultural heritage.
- **Requirements:** For a virtual research environment for cultural heritage, we need
  - scientific communication about and documentation of the cultural heritage
  - networking knowledge from different disciplines (transdisciplinarity)
  - high-quality data acquisition and analysis
  - safeguarding authorship, authenticity, persistence
  - support of scientific publication
- WissKI was developed by the research group of Prof. Günther Görtz at FAU Erlangen-Nürnberg and is now used in hundreds of DH projects across Germany.
- FAU supports cultural heritage research by providing hosted WissKI instances.
  - See `https://wisski.data.fau.de` for details
  - We will use an instance for the Kirmes paintings in the homework assignments

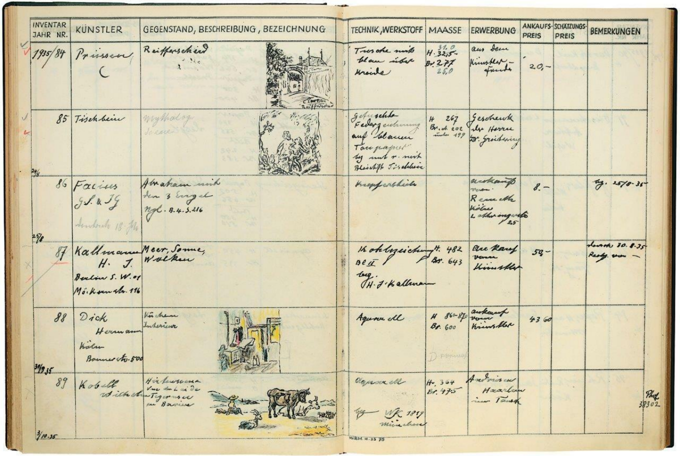
Documenting Cultural Heritage: Current State/Preview

- Pre-DH State of cultural heritage documentation:
  - **scientific communication/documentation** by journal articles/books
  - **persistence**: paper records, file cards, databases (like our KirmesDB)
  - **Analysis**: manual examination of artefacts in museums/archives.
- **Idea:** Use more technology to do better.
- **Preview:** WissKI uses semantic web technologies to do just that. We will now
  - Motivate the semantic web (why do we need more than the WWW)
  - introduce ontologies, linked open data and their technology stacks
  - show off WissKI and offer a little project based on Kirmes corpus.

12.2 Systems for Documenting the Cultural Heritage

Documenting Cultural Artefacts: Inventory Books

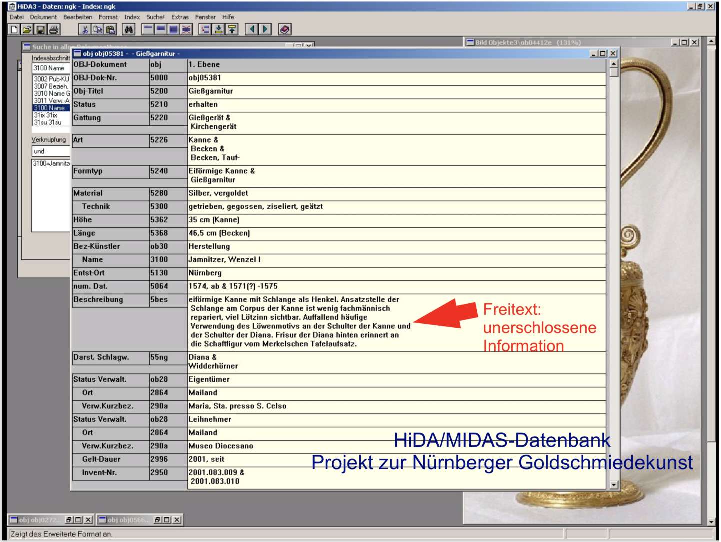
- **Definition 2.1.** An inventory book is a ledger that identifies, describes, and records provenance of the artefacts in the collection of a museum.
- **Example 2.2 (An Inventory Book).**



- **Problems:** non-digital, only single-user access, institution-local, no querying, ...

Cultural Artefacts in Databases: Example

- **Example 2.3.** A typical database for cultural artefacts: (HiDa/MIDAS)

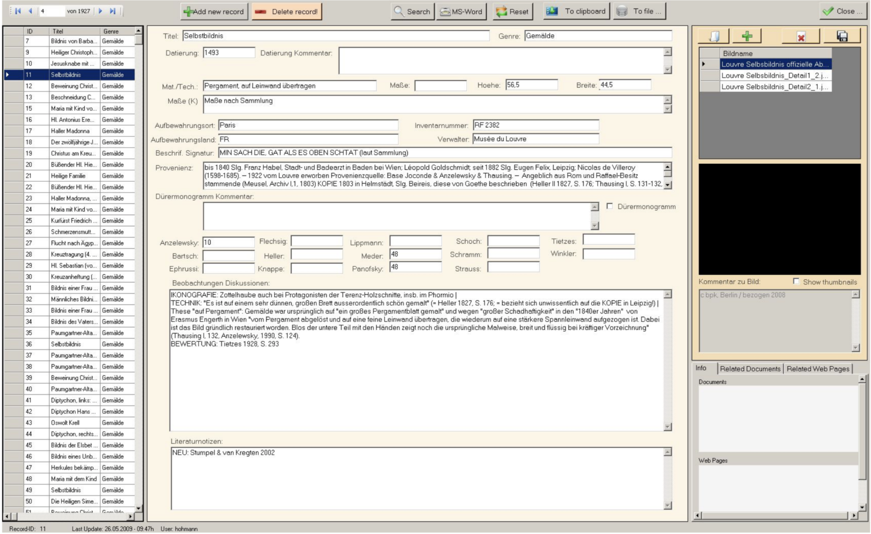


Cultural Artefacts in Databases: Pro/Con

- **Databases of Cultural Artefacts – Advantages:**
  - persistence, multi-user access, structured data,
  - web/catalog publication, standardized exports,
  - standardized performant query language.
- **Databases of Cultural Artefacts – Problems:**
  - identifiers are database local $\rightsquigarrow$ no trans database relations,
  - database schemata are inflexible $\leftrightsquigarrow$ we need extensions in practice,
  - free text as an un-structured, untapped resource.
- **Idea:** Relational databases impose structure, let's try something very unstructured: the world wide web. (up next)

Cultural Artefacts in Databases II

- **Example 2.4.** Another database for cultural artefacts:



Using the Web for the Cultural Heritage

- **Idea:** Why not use the world wide web as a tool?
  - it is inherently distributed and networked,
  - the data formats HTML and XML are highly flexible,
  - gives us instantaneous access to information/images/...,
  - allows collaboration and discussion. (wikis, fora, blogs)

Cultural Artefacts on the Web

- **Example 2.5.** A text about a cultural artefact (an etching by Dürer)

Not logged in Talk Contributions Create account Log in

Article Talk Read Edit View history Search Wikipedia

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Current events
Random article
About Wikipedia
Contact us
Donate

Contribute
Help
Community portal
Recent changes
Upload file

Tools
What links here
Related changes
Special pages
Permanent link
Page information
Cite this page
Wikidata item

Print/export

Melencolia I

From Wikipedia, the free encyclopedia

Melencolia I is a 1514 engraving by the German Renaissance artist Albrecht Dürer. The print's central subject is an enigmatic and gloomy winged female figure thought to be a personification of melancholia. Holding her head in her hand, she stares past the busy scene in front of her. The area is strewn with symbols and tools associated with craft and carpentry, including an hourglass, weighing scales, a hand plane, a claw hammer, and a saw. Other objects relate to alchemy, geometry or numerology. Behind the figure is a structure with an embedded magic square, and a ladder leading beyond the frame. The sky contains a rainbow, a comet or planet, and a bat-like creature bearing the text that has become the print's title.

Dürer's engraving is one of the most well-known extant old master prints, but, despite a vast art-historical literature, it has resisted any definitive interpretation. Dürer may have associated melancholia with creative activity;[2] the woman may be a representation of a Muse, awaiting inspiration but fearful that it will not return. As such, Dürer may have intended the print as a veiled self-portrait. Other art historians see the figure as pondering the nature of beauty or the value of artistic creativity in light of rationalism,[3] or as a purposely obscure work that highlights the limitations of allegorical or symbolic art.

The art historian Erwin Panofsky, whose writing on the print has received the

Melencolia I

Melencolia I[1] (with annotations)

Artist	Albrecht Dürer
Year	1514
Type	engraving
Dimensions	24 cm × 18.8 cm (9.4 in × 7.4 in)

- **Question:** Just how does the etching discussed here relate to Albrecht Dürer?

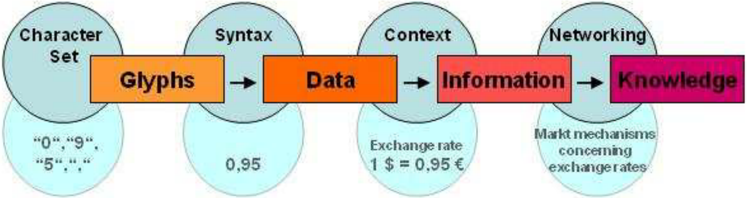
Using the Web for Cultural Heritage

- **Problems:** with using the Web as a resource
  - Information is often of dubious quality (imprecise, typos, incomplete, . . .)
  - Information is primarily written for human consumption
    - $\leadsto$ not machine-actionable, but full text search works (e.g. Google)
    - sometimes we can use established structures (e.g. Infobox in Wikipedia)
- **Evaluation:** The web is complementary to databases on the structure-vs-flexibility tradeoff scale for cultural heritage systems. (we need both)
- **Idea:** Use the semantic web for cultural heritage
  - **Goal**: Make information accessible for humans and machines
  - meaning capture by reference to real-world objects
  - globally unique identifiers of cultural artefacts ($\widehat{=}$ URIs)
  - inference (get out more than you put in!)

12.3 The Semantic Web

The Semantic Web

- **Definition 3.1.** The semantic web is the result including of semantic content in web pages with the aim of converting the WWW into a machine-understandable "web of data", where inference based services can add value to the ecosystem.
- **Idea:** Move web content up the ladder, use inference to make connections.



- **Example 3.2.** Information not explicitly represented (in one place)

  Query: *Who was US president when Barak Obama was born?*
  Google: *. . . BIRTH DATE: August 04, 1961. . .*
  Query: *Who was US president in 1961?*
  Google: *President: Dwight D. Eisenhower [. . .] John F. Kennedy (starting Jan. 20.)*

  Humans understand the text and combine the information to get the answer. Machines need more than just text $\leadsto$ semantic web technology.

What is the Information a User sees?

- **Example 3.3.** Take the following web-site with a conference announcement

  WWW2002
  The eleventh International World Wide Web Conference
  Sheraton Waikiki Hotel
  Honolulu, Hawaii, USA
  7-11 May 2002
  Registered participants coming from
  Australia, Canada, Chile Denmark, France, Germany, Ghana, Hong Kong, India, Ireland, Italy, Japan, Malta, New Zealand, The Netherlands, Norway, Singapore, Switzerland, the United Kingdom, the United States, Vietnam, Zaire

  On the 7th May Honolulu will provide the backdrop of the eleventh International World Wide Web Conference.

  Speakers confirmed
  Tim Berners-Lee: Tim is the well known inventor of the Web,
  Ian Foster: Ian is the pioneer of the Grid, the next generation internet.

What the machine sees

- **Example 3.4.** Here is what the machine “sees” from the conference announcement:

  𝒲𝒲𝒲∈″∈

  𝒯⟨]]↕]⊑]\⊔⟨ℐ\⊔]∇\⊣⊔⟩≀\⊣↕𝒲≀∇↕⌈𝒲⟩⌈]𝒲]⌊𝒞≀\{]∇]\⌋]

  𝒮⟨]∇⊣⊔≀\𝒲⊣⟩∥⟩∥⟩ℋ≀⊔]↕

  ℋ≀\≀↕⊓↕⊓⇔ℋ⊣⊒⊣⟩⟩⇔𝒰𝒮𝒜

  ↖∞∞ℳ⊣†∈″∈

  ℛ]}⟩∫⊔]∇]⌈√⊣∇⊔⟩⌋⟩√⊣\⊔∫]≀⇕⟩\}{∇≀⇕

  𝒜⊓∫⊔∇⊣↕⟩⊣⇔𝒞⊣\⊣⌈⊣⇔𝒞⟨⟩↕]𝒟]\⇕⊣∇∥⇔ℱ∇⊣\⌋]⇔𝒢]∇⇕⊣\†⇔𝒢⟨⊣\⊣⇔ℋ≀\}𝒦≀\}⇔ℐ\⌈⟩⊣⇔

  ℐ∇]↕⊣\⌈⇔ℐ⊔⊣↕†⇔𝒥⊣√⊣\⇔ℳ⊣↕⊔⊣⇔𝒩]⊒𝒵]⊣↕⊣\⌈⇔𝒯⟨]𝒩]⊔⟨]∇↕⊣\⌈∫⇔𝒩≀∇⊒⊣†⇔

  𝒮⟩\}⊣√≀∇]⇔𝒮⊒⟩⊔‡]∇↕⊣\⌈⇔⊔⟨]𝒰\⟩⊔]⌈𝒦⟩\}⌈≀⇕⇔⊔⟨]𝒰\⟩⊔]⌈𝒮⊔⊣⊔]∫⇔𝒱⟩]⊔\⊣⇕⇔𝒵⊣⟩∇]

  𝒪\⊔⟨]⊔⟨ℳ⊣†ℋ≀\≀↕⊓↕⊓⊒⟩↕↕√∇≀⊑⟩⌈]⊔⟨]⌊⊣]∥⌈∇≀√≀{⊔⟨]]↕]⊑]\⊔⟨

  ℐ\⊔]∇\⊣⊔⟩≀\⊣↕𝒲≀∇↕⌈𝒲⟩⌈]𝒲]⌊𝒞≀\{]∇]\⌋]↙

  𝒮√]⊣∥]∇∫]≀\{⟩∇⇕]⌈

  𝒯⟩⇕ℬ]∇\]∇∫↖ℒ]]¬𝒯⟩⇕⟩∫⊔⟨]⊒]↕↕∥\≀⊒\⟩\⊑]\⊔≀∇≀{⊔⟨]𝒲]⌊⇔

  ℐ⊣\ℱ≀∫⊔]∇¬ℐ⊣\⟩∫⊔⟨]√⟩≀\]]∇≀{⊔⟨]𝒢∇⟩⌈⇔⊔⟨]\]§⊔}]\]∇⊣⊔⟩≀\⟩\⊔]∇\]⊔↙

Solution: XML markup with "meaningful" Tags

- **Example 3.5.** Let's annotate (parts of) the meaning via XML markup

  <title>𝒲𝒲𝒲∈′′∈
  𝒯⟨]]↕]⊑]\⊔⟨ℐ\⊔]∇\⊣⊔⟩≀\⊣↕𝒲≀∇↕[𝒲⟩⌈]𝒲]⌊𝒞≀\{]∇]\⌋] </title>
  <place>𝒮⟨]∇⊣⊔≀\𝒲⊣⟩∥⟩∥⟩ℋ≀⊔]↕ℋ≀\≀↕⊓↕⊓⇔ℋ⊣⊒⊣⟩⟩⇔𝒰𝒮𝒜</place>
  <date>↖∞∞ℳ⊣†∈′′∈</date>
  <participants>ℛ]}⟩∫⊔]∇]⌈√⊣∇⊔⟩⌋⟩√⊣\⊔∫]≀↕⟩\}{∇≀↕
  𝒜⊓∫⊔∇⊣↕⟩⊣⇔𝒞⊣\⊣⌈⊣⇔𝒞⟨⟩↕]𝒟]\↕⊣∇∥⇔ℱ∇⊣\⌋]⇔𝒢]∇↕⊣\†⇔𝒢⟨⊣\⊣⇔ℋ≀\}𝒦≀\}⇔ℐ\⌈⟩⊣⇔
  ℐ∇]↕⊣\⌈⇔ℐ⊔⊣↕†⇔𝒥⊣√⊣\⇔ℳ⊣↕⊔⊣⇔𝒩]⊒𝒵]⊣↕⊣\⌈⇔𝒯⟨]𝒩]⊔⟨]∇↕⊣\⌈∫⇔𝒩≀∇⊒⊣†⇔
  𝒮⟩\}⊣√≀∇]⇔𝒮⊒⟩⊔‡]∇↕⊣\⌈⇔⊔⟨]𝒰\⟩⊔]⌈𝒦⟩\}⌈≀↕⇔⊔⟨]𝒰\⟩⊔]⌈𝒮⊔⊣⊔]∫⇔𝒱⟩]⊔\⊣↕⇔𝒵⊣⟩∇]
  </participants>
  <introduction>𝒪\⊔⟨]⊔⟨ℳ⊣†ℋ≀\≀↕⊓↕⊓⊒⟩↕↕√∇≀⊑⟩⌈]⊔⟨]⌊⊣]∥⌈∇≀√≀{⊔⟨]]↕]⊑]\⊔⟨ℐ\↖
  ⊔]∇\⊣⊔⟩≀\⊣↕𝒲≀∇↕[𝒲⟩⌈]𝒲]⌊𝒞≀\{]∇]\⌋]↙</introduction>
  <program>𝒮√]⊣∥]∇∫]≀\{⟩∇↕]⌈
  <speaker>𝒯⟩↕ℬ]∇\]∇∫↖ℒ]]¬𝒯⟩↕⟩∫⊔⟨]⊒]↕↕∥\≀⊒\⟩\⊑]\⊔≀∇≀{⊔⟨]𝒲]⌊</speaker>
  <speaker>ℐ⊣\ℱ≀∫⊔]∇¬ℐ⊣\⟩∫⊔⟨]√⟩≀\]]∇≀{⊔⟨]𝒢∇⟩⌈⇔⊔⟨]\]§⊔}]\]∇⊣⊔⟩≀\⟩\⊔]∇↖
  \]⊔<speaker>
  </program>

What can we do with this?

- **Example 3.6.** Consider the following fragments:

  ℜ⊔⟩⊔↕]⊤𝒲𝒲𝒲∈′′∈

  𝒯⟨]]↕]⊑]\⊔⟨ℐ\⊔]∇\⊣⊔⟩≀\⊣↕𝒲≀∇↕⌈𝒲⟩⌈]𝒲]⌊𝒞≀\{]∇]\⌋]ℜ∝⊔⟩⊔↕]⊤

  ℜ√↕⊣⌋]⊤𝒮⟨]∇⊣⊔≀\𝒲⊣⟩∥⟩∥⟩ℋ≀⊔]↕ℋ≀\≀↕⊓↕⊓⇔ℋ⊣⊒⊣⟩⟩⇔𝒰𝒮𝒜ℜ∝√↕⊣⌋]⊤

  ℜ⌈⊣⊔]⊤↖∞∞ℳ⊣†∈′′∈ℜ∝⌈⊣⊔]⊤

  Given the markup above, a machine agent can
  - parse ∞∞ℳ⊣†∈′′∈ as the date May 7 11 2002 and add this to the user's calendar,
  - parse 𝒮⟨]∇⊣⊔≀\𝒲⊣⟩∥⟩∥⟩ℋ≀⊔]↕ℋ≀\≀↕⊓↕⊓⇔ℋ⊣⊒⊣⟩⟩⇔𝒰𝒮𝒜 as a destination and find flights.
- **But:** do not be deceived by your ability to understand English!

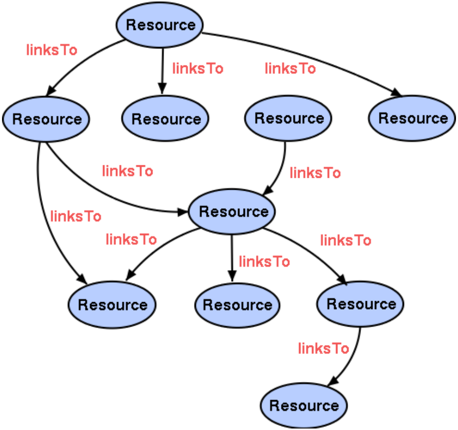
What the machine sees of the XML

- **Example 3.7.** Here is what the machine sees of the XML

  <title>WWW∈′′∈
  T⟨]]↕]⊑]\⊔⟨I\⊔]∇\⊣⊔⟩≀\⊣↕W≀∇↕⌈W⟩⌈]W]⌊C≀\{]∇]\⌋] </⊔⟩⊔↕]>
  < √↕⊣⌋]>S⟨]∇⊣⊔≀\W⊣⟩∥⟩∥⟩H≀⊔]↕H≀\≀↕⊓↕⊓⇔H⊣⊒⊣⟩⟩⇔USA</ √↕⊣⌋]>
  <⌈⊣⊔]>↖∞∞M⊣†∈′′∈</⌈⊣⊔]>
  < √⊣∇⊔⟩⌋⟩ √⊣\⊔∫>R]}⟩∫⊔]∇]⌈ √⊣∇⊔⟩⌋⟩ √⊣\⊔∫⌋≀⇕⟩\}{∇≀⇕
  A⊓∫⊔∇⊣⇕⟩⊣⇔C⊣\⊣⌈⊣⇔C⟨⟩↕]D]\⇕⊣∇∥⇔F∇⊣\⌋]⇔G]∇⇕⊣\†⇔G⟨⊣\⊣⇔H≀\}K≀\}⇔I\⌈⟩⊣⇔
  I∇]↕⊣\⌈⇔I⊔⊣⇕†⇔J⊣ √⊣\⇔M⊣⇕⊔⊣⇔N]⊒Z]⊣⇕⊣\⌈⇔T⟨]N]⊔⟨]∇↕⊣\⌈∫⇔N≀∇⊒⊣†⇔
  S⟩\}⊣ √≀∇]⇔S⊒⟩⊔‡]∇↕⊣\⌈⇔⊔⟨]U\⟩⊔]⌈K⟩\}⌈≀⇕⇔⊔⟨]U\⟩⊔]⌈S⊔⊣⊔]∫⇔V⟩]⊔\⊣⇕⇔Z⊣⟩∇]
  </ √⊣∇⊔⟩⌋⟩ √⊣\⊔∫>
  <⟩\⊔∇≀⌈⊓⌋⊔⟩≀\>O\⊔⟨]⊔⟨M⊣†H≀\≀↕⊓↕⊓⊒⟩↕↕ √∇≀⊑⟩⌈]⊔⟨]⌊⊣⌋∥⌈∇≀ √≀{⊔⟨]]↕]⊑]\⊔⟨I\⊔]∇↖
  \⊣⊔⟩≀\⊣↕W≀∇↕⌈W⟩⌈]W]⌊C≀\{]∇]\⌋]↙</⟩\⊔∇≀⌈⊓⌋⊔⟩≀\>
  < √∇≀}∇⊣⇕>S √]⊣∥]∇∫⌋≀\{⟩∇⇕]⌈
  <∫ √]⊣∥]∇>T⟩⇕B]∇]\]∇∫↖L]]¬T⟩⇕⟩∫⊔⟨]⊒]↕↕∥\≀⊒\⟩\⊑]\⊔≀∇≀{⊔⟨]W]⌊</∫ √]⊣∥]∇>
  <∫ √]⊣∥]∇>I⊣\F≀∫⊔]∇¬I⊣\⟩∫⊔⟨] √⟩≀\]]∇≀{⊔⟨]G∇⟩⌈⇔⊔⟨]\]§⊔}]\]∇⊣⊔⟩≀\⟩\⊔]∇↖
  \]⊔<∫ √]⊣∥]∇>
  </ √∇≀}∇⊣⇕>

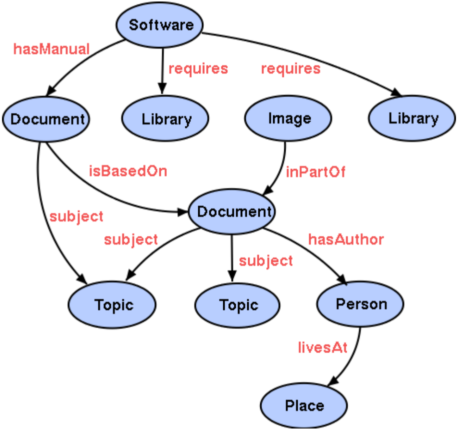
The Current Web

- **Resources:** identified by URIs, untyped
- **Links:** href, src, ...limited, non-descriptive
- **User:** Exciting world - semantics of the resource, however, gleaned from content
- **Machine:** Very little information available - significance of the links only evident from the context around the anchor.



The Semantic Web

- **Resources:** Globally identified by URIs or Locally scoped (Blank), Extensible, Relational.
- **Links:** Identified by URIs, Extensible, Relational.
- **User:** Even more exciting world, richer user experience.
- **Machine:** More processable information is available (Data Web).
- **Computers and people:** Work, learn and exchange knowledge effectively.



Towards a "Machine-Actionable Web"

- **Recall:** We need external agreement on meaning of annotation tags.
- **Idea:** standardize them in a community process (e.g. DIN or ISO)
- **Problem:** Inflexible, Limited number of things can be expressed

Towards a “Machine-Actionable Web”

- **Recall:** We need external agreement on meaning of annotation tags.
- **Idea:** standardize them in a community process (e.g. DIN or ISO)
- **Problem:** Inflexible, Limited number of things can be expressed
- **Better:** Use ontologies to specify meaning of annotations
  - Ontologies provide a vocabulary of terms
  - New terms can be formed by combining existing ones
  - Meaning (semantics) of such terms is formally specified
  - Can also specify relationships between terms in multiple ontologies

Towards a "Machine-Actionable Web"

- **Recall:** We need external agreement on meaning of annotation tags.
- **Idea:** standardize them in a community process (e.g. DIN or ISO)
- **Problem:** Inflexible, Limited number of things can be expressed
- **Better:** Use ontologies to specify meaning of annotations
  - Ontologies provide a vocabulary of terms
  - New terms can be formed by combining existing ones
  - Meaning (semantics) of such terms is formally specified
  - Can also specify relationships between terms in multiple ontologies
- Inference with annotations and ontologies (get out more than you put in!)
  - Standardize annotations in RDF [KC04] or RDFa [Her+13] and ontologies on OWL [OWL09]
  - Harvest RDF and RDFa in to a triplestore or OWL reasoner.
  - Query that for implied knowledge (e.g. chaining multiple facts from Wikipedia)

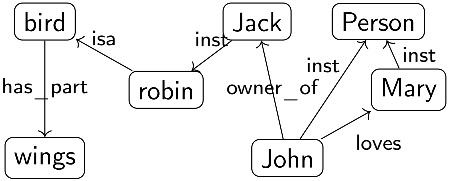
    SPARQL: Who was US President when Barack Obama was Born?

    DBPedia: John F. Kennedy (was president in August 1961)

12.4 Semantic Networks and Ontologies

Semantic Networks [CQ69]

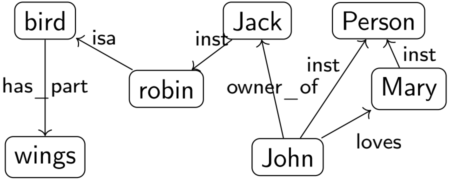
- **Definition 4.1.** A semantic network is a directed graph for representing knowledge:
  - nodes represent objects and concepts (classes of objects) (e.g. John (object) and bird (concept))
  - edges (called links) represent relations between these (isa, father_of, belongs_to)
- **Example 4.2.** A semantic network for birds and persons:



- **Problem:** How do we derive new information from such a network?
- **Idea:** Encode taxonomic information about objects and concepts in special links (“isa” and “inst”) and specify property inheritance along them in the process model.

Deriving Knowledge Implicit in Semantic Networks

- **Observation 4.3.** *There is more knowledge in a semantic network than is explicitly written down.*
- **Example 4.4.** In the network below, we "know" that *robins have wings* and in particular, *Jack has wings*.



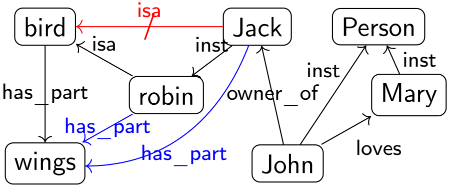
- **Idea:** Links labeled with "isa" and "inst" are special: they propagate properties encoded by other links.
- **Definition 4.5.** We call links labeled by
  - "isa" an inclusion or isa link (inclusion of concepts)
  - "inst" instance or inst link (concept membership)

Deriving Knowledge Semantic Networks

- **Definition 4.6 (Inference in Semantic Networks).** We call all link labels except "inst" and "isa" in a semantic network relations.

  Let N be a semantic network and R a relation in N such that $A \xrightarrow{\text{isa}} B \xrightarrow{R} C$ or $A \xrightarrow{\text{inst}} B \xrightarrow{R} C$, then we can derive a relation $A \xrightarrow{R} C$ in N.

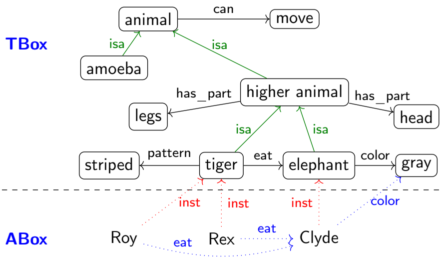
  The process of deriving new concepts and relations from existing ones is called inference and concepts/relations that are only available via inference implicit (in a semantic network).
- **Intuition:** Derived relations represent knowledge that is implicit in the network; they could be added, but usually are not to avoid clutter.
- **Example 4.7.** Derived relations in 4.4



- **Slogan:** Get out more knowledge from a semantic networks than you put in.

Terminologies and Assertions

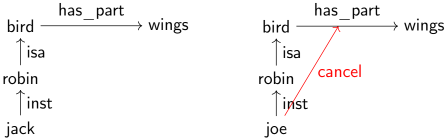
- *Remark 4.8.* We should distinguish concepts from objects.
- **Definition 4.9.** We call the subgraph of a semantic network N spanned by the isa links and relations between concepts the terminology (or TBox, or the famous Isa Hierarchy) and the subgraph spanned by the inst links and relations between objects, the assertions (or ABox) of N.
- **Example 4.10.** In this semantic network we keep objects concept apart notationally:



In particular we have objects “Rex”, “Roy”, and “Clyde”, which have (derived) relations (e.g. *Clyde* is *gray*).

Limitations of Semantic Networks

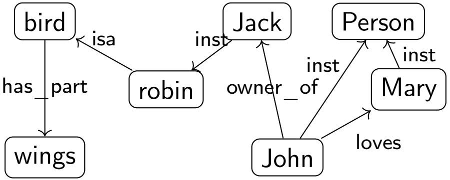
- What is the meaning of a link?
  - link labels are very suggestive (misleading for humans)
  - meaning of link types defined in the process model (no denotational semantics)
- **Problem:** No distinction of optional and defining traits!
- **Example 4.11.** Consider a robin that has lost its wings in an accident:



"Cancel-links" have been proposed, but their status and process model are debatable.

Another Notation for Semantic Networks

- **Definition 4.12.** Function/argument notation for semantic networks
  - interprets nodes as arguments (reification to individuals)
  - interprets links as functions (predicates actually)
- **Example 4.13.**

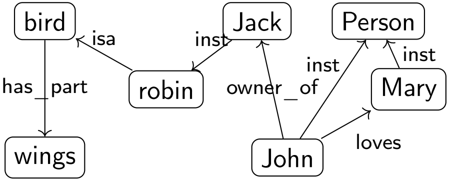


isa(robin,bird)
haspart(bird,wings)
inst(Jack,robin)
owner_of(John, robin)
loves(John,Mary)

- **Evaluation:**
  - \+ linear notation (equivalent, but better to implement on a computer)
  - \+ easy to give process model by deduction (e.g. in Prolog)
  - − worse locality properties (networks are associative)

A Denotational Semantics for Semantic Networks

- **Observation:** If we handle isa and inst links specially in function/argument notation



robin $\subseteq$ bird
haspart(bird,wings)
Jack$\in$robin
owner_of(John, Jack)
loves(John,Mary)

it looks like first-order logic, if we take
  - $a \in S$ to mean $S(a)$ for an object a and a concept S.
  - $A \subseteq B$ to mean $\forall X.A(X) \Rightarrow B(X)$ and concepts A and B
  - $R(A, B)$ to mean $\forall X.A(X) \Rightarrow (\exists Y.B(Y) \wedge R(X, Y))$ for a relation R.
- **Idea:** Take first-order deduction as process model (gives inheritance for free)

What is an Ontology

- **Definition 4.14.** An ontology is a formal model of (an aspect of) the world. It
  - introduces a vocabulary for the objects, concepts, and relations of a given domain,
  - specifies intended meaning of vocabulary in a description logic using
    - a set of axioms describing structure of the model
    - a set of facts describing some particular concrete situation

  The vocabulary together with the collection of axioms is often called a terminology (or TBox) and the collection of facts an ABox (assertions).
  In addition to the represented axioms and facts, the description logic determines a number of derived ones.
- **Definition 4.15.** A vocabulary often includes names for classes and relationship (also called concepts, and properties).
- *Remark 4.16.* If the description logic has a reasoner, we can automatically
  - detect inconsistent axiom systems
  - compute class membership and taxonomies.

Semantic Web Technology in a Nutshell

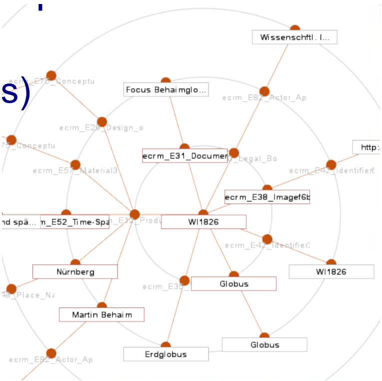
- Ontologies have become one of the standard devices for representing information about the Web and the world.
- **Definition 4.17.** This is facilitated and standardized by the :
  - URIs for representing objects,
  - RDF triples for representing facts,
  - RDFa for annotating RDF triples in XML documents,
  - OWL for representing TBoxes,
  - triplestores for storing (lots of) RDF triples,
  - SPARQL for querying ontologies,
  - description logic reasoners for deciding ontology consistency and concept subsumption,
  - Protg for authoring and maintaining ontologies,
- Details .

12.5 CIDOC CRM: An Ontology for Cultural Heritage

Ontologies for Cultural Artefacts

- **Idea:** Use ontologies for documenting cultural heritage.
  - flexible schemata (OWL)
  - easy data sharing
  - open standards, free tools
  - semantic querying via SPARQL
- **Idea:** We can use RDF like a Mindmap: RDF can
  - represent relations between objects
  - classify objects (web resources)

  RDFa for document annotation



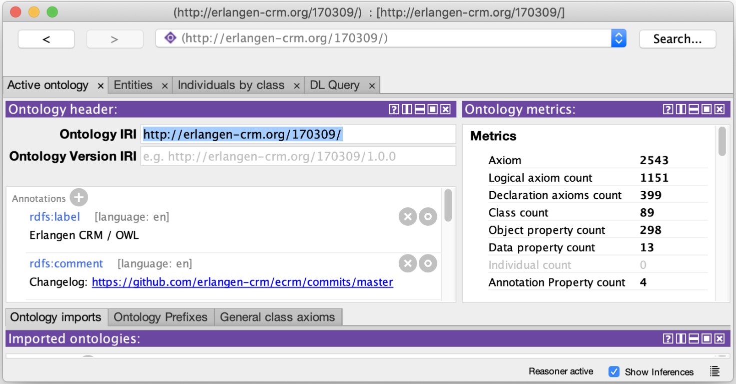
- Reference ontologies for interoperability:
  - SUMO (Suggested Upper Model Ontology) [SUMO] for common knowledge,
  - FOAF (Friend-of-a-Friend) [FOAF14] for persons and relations,
  - CIDOC CRM for documentation of cultural heritage. (up next)

CIDOC CRM (Conceptual Reference Model)

- **Definition 5.1.** CIDOC CRM provides an extensible ontology for concepts and information in cultural heritage and museum documentation. It is the international standard (ISO 21127:2014) for the controlled exchange of cultural heritage information. The central classes include
  - space time specified by title/identifier, place, era/period, time-span, and relationship to persistent items
  - events specified by title/identifier, beginning/ending of existence, participants (people, either individually or in groups), creation/modification of things (physical or conceptional), and relationship to persistent items
  - material things specified by title/identifier, place, the information object the material thing carries, part-of relationships, and relationship to persistent items
  - immaterial things specified by title/identifier, information objects (propositional or symbolic), conceptional things, and part-of relationships
- **Definition 5.2.** OWL implements CIDOC CRM in OWL
- Details about CIDOC CRM can be found at [CC] and about OWL at [ECRMb; ECRMa].

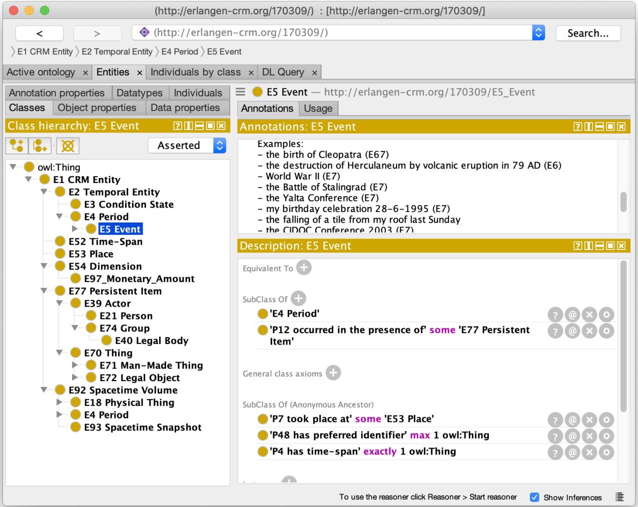
Protege, an IDE for Ontology Development

- **Definition 5.3.** Protg [Pro] is an integrated development environment for ontologies represented in the OWL family. It comprises
  - a visual user interface for exploring and editing ontologies,
  - a inference component to ensure ontology consistency and minimality,
  - a facility for querying the loaded ontologies.
- **Example 5.4 (CIDOCCRM in Protege).**



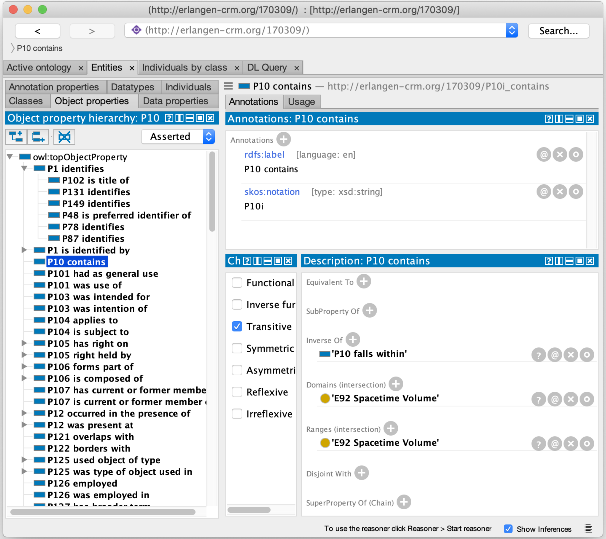
CIDOC CRM Explored (Classes)

- **Idea:** Use semantic web technology to explore OWL.
- **CIDOC CRM Classes:** concept $\widehat{=}$ OWL “Class” (shown in Protege)

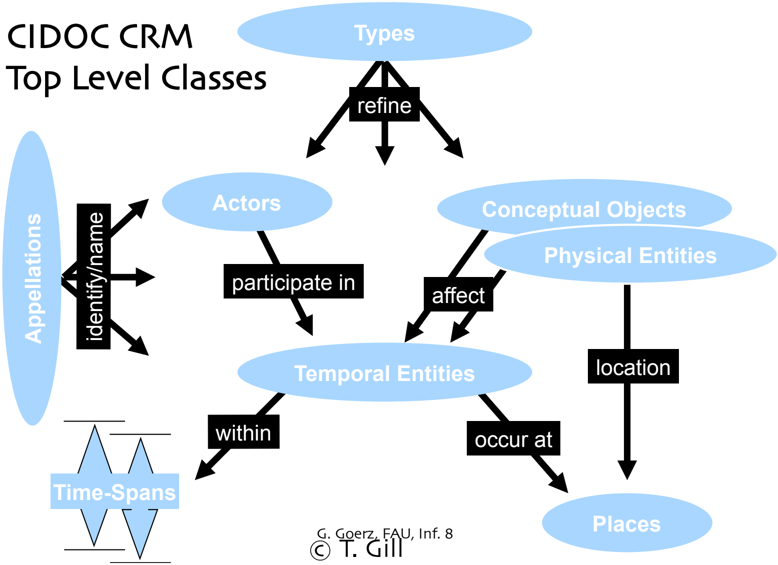


CIDOC CRM Explored (Relations)

- **CIDOC CRM Relations:** relation $\hat{=}$ OWL "Object Property" (shown in Protege)



CIDOC CRM Structure (Overview)



CIDOC-CRM Modeling

- **This is all good and dandy** but how do I concretely model cultural artefacts?
- **Answer:** CIDOC CRM is only a TBox, we add an ABox of objects and facts.
- **Example 5.5.** *Albrecht Dürer painted Melencolia 1 in Nürnberg*
  We have two units of information here:
  1. Albrecht Dürer painted Melencolia 1
  2. this happened in the city of Nürnberg
- CIDOC CRM modeling decisions; we start with 1. *AD painted M 1*
  1. A painting *m* is an "Information Carrier" (E84)
  2. It was created in an "Production Event" *q* (E12)
  3. *m* is related to *q* via the "was produced by" relation (P108i)
  4. *q* was "carried out by" a "person" *d* (P14 E21)
  5. *d* "is identified by" an "actor appellation" *a* (P131 E82)
  6. *a* "has note" the string "Albrecht Dürer". (P3)
- CIDOC CRM modeling decisions; continuing with 2. *this happened in N*
  1. A painting *m* is an "Information Carrier" (E84)
  2. It was created in an "Production Event" *q* (E12)
  3. *m* is related to *q* via the " produced by" relation (P108i)
  4. *q* "took place at" a "place" *p* (P7 E53)
  5. *p* "is identified by" a "place name" *n* (P48 E3)
  6. *n* "has note" the string "Nürnberg". (P3)

CIDOC CRM Modelling (Ontology Paths)

- Modeling *Albrecht Dürer painted Melencolia 1 in Nürnberg* in CIDOC CRM

$$
\begin{tikzcd}
 & & d : E21 \arrow[r, "\mathrm{P131}"] & a : E82 \arrow[r, "\mathrm{P3}"] & \texttt{"A. Dürer"} \\
m : E84 \arrow[r, "\mathrm{P108i}"] & q : E12 \arrow[ur, "\mathrm{P14}"] \arrow[dr, "\mathrm{P7}"'] & & & \\
 & & p : E53 \arrow[r, "\mathrm{P87}"] & n : E48 \arrow[r, "\mathrm{P3}"] & \texttt{"Nürnberg"}
\end{tikzcd}
$$

  Note that we need to create the intermediary objects q, d, a, and n.
- **Problem:** That is a lot of work for something very simple.
- **Definition 5.6.** We call sequence of facts $s_i \xrightarrow{p_i} o_i$, where $s_i = o_{i-1}$ an ontology path and any subtree an ontology group.
- **Problem Reformulated:** A simple statement like *Albrecht Dürer painted Melencolia 1* becomes a whole ontology path in CIDOC CRM.
- **But:** we can reuse intermediary objects and facts, and need fine grained models for flexibility.
- **Idea:** Maybe systems can take some of the pain out of modeling. ($\leadsto$ WissKI)

Event-Oriented Modeling in CIDOC CRM

- **Observation 5.7.** *Ontologies make it easy to model facts with transitive verbs, e.g. Albrecht Dürer created Melencolia 1* *(binary relation)*
- **Problem:** What about more complex situations with more arguments? E.g.
  1. *Albrecht Dürer created Melencolia 1 with an etching needle* (ternary)
  2. *Albrecht Dürer created Melencolia 1 with an etching needle in Nürnberg* (four arguments)
  3. *Albrecht Dürer created Melencolia 1 with an etching needle in Nürnberg out of boredom* (five)
- **Standard Solution:** Introduce "events" tied to the verb and describe those
- **Example 5.8.** There was a creation event *e* with
  1. *Albrecht Dürer* as the agent,
  2. *Melencolia 1* as the product,
  3. *an etching needle* as the means,
  4. *boredom* as the reason,
- **Consequence:** More than 1/3 of CIDOC CRM classes are events of some kind.

12.6 The Semantic Web Technology Stack

Resource Description Framework

- **Definition 6.1.** The Resource Description Framework (RDF) is a framework for describing resources on the web. It is an XML vocabulary developed by the W3C.
- **Note:** RDF is designed to be read and understood by computers, not to be displayed to people. (it shows)
- **Example 6.2.** RDF can be used for describing (all "objects on the WWW")
  - properties for shopping items, such as price and availability
  - time schedules for web events
  - information about web pages (content, author, created and modified date)
  - content and rating for web pictures
  - content for search engines
  - electronic libraries

Resources and URIs

- RDF describes resources with properties and property values.
- RDF uses Web identifiers (URIs) to identify resources.
- **Definition 6.3.** A resource is anything that can have a URI, such as `http://www.fau.de`.
- **Definition 6.4.** A property is a resource that has a name, such as *author* or *homepage*, and a property value is the value of a property, such as *Michael Kohlhase* or `http://kwarc.info/kohlhase`. (a property value can be another resource)
- **Definition 6.5.** A RDF statement s (also known as a triple) consists of a resource (the subject of s), a property (the predicate of s), and a property value (the object of s). A set of RDF triples is called an RDF graph.
- **Example 6.6.** Statements: *[This slide]^{subj} has been [author]^{pred}ed by [Michael Kohlhase]^{obj}*

XML Syntax for RDF

- RDF is a concrete XML vocabulary for writing statements
- **Example 6.7.** The following RDF document could describe the slides as a resource

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22−rdf−syntax−ns#"
         xmlns:dc= "http://purl.org/dc/elements/1.1/">
  <rdf:Description about="https://.../CompLog/kr/en/rdf.tex">
    <dc:creator>Michael Kohlhase</dc:creator>
    <dc:source>http://www.w3schools.com/rdf</dc:source>
  </rdf:Description>
</rdf:RDF>
```

  This RDF document makes two statements:
  - The subject of both is given in the about attribute of the rdf:Description element
  - The predicates are given by the element names of its children
  - The objects are given in the elements as URIs or literal content.
- **Intuitively:** RDF is a web-scalable way to write down ABox information.

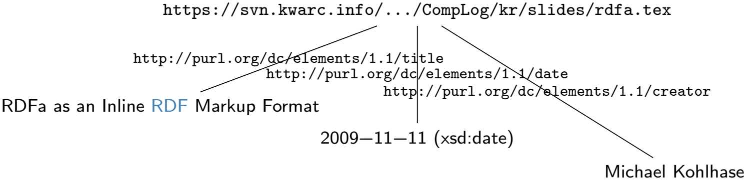
RDFa as an Inline RDF Markup Format

- **Problem:** RDF is a standoff markup format (annotate by URIs pointing into other files)

  Definition 6.8. RDFa (RDF annotations) is a markup scheme for inline annotation (as XML attributes) of RDF triples.

- **Example 6.9.**

```
<div xmlns:dc="http://purl.org/dc/elements/1.1/" id="address">
    <h2 about="#address" property="dc:title">RDF as an Inline RDF Markup Format</h2>
    <h3 about="#address" property="dc:creator">Michael Kohlhase</h3>
    <em about="#address" property="dc:date" datatype="xsd:date"
        content="2009−11−11">November 11., 2009</em>
</div>
```



RDF as an ABox Language for the Semantic Web

- **Idea:** RDF triples are ABox entries h R s or $h{:}\varphi$.
- **Example 6.10.** h is the resource for Ian Horrocks, s is the resource for Ulrike Sattler, R is the relation "hasColleague", and φ is the class foaf:Person

```
<rdf:Description about="some.uri/person/ian_horrocks">
  <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
  <hasColleague resource="some.uri/person/uli_sattler"/>
</rdf:Description>
```

- **Idea:** Now, we need an similar language for TBoxes (based on $\mathcal{ALC}$)

OWL as an Ontology Language for the Semantic Web

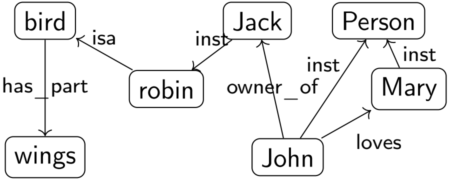
- **Task:** Complement RDF (ABox) with a TBox language.
- **Idea:** Make use of resources that are values in rdf:type. (called Classes)
- **Definition 6.11.** OWL (the ontology web language) is a language for encoding TBox information about RDF classes.
- **Example 6.12 (A concept definition for "Mother").**
  Mother=Woman ⊓ Parent is represented as

XML Syntax	Functional Syntax

`<EquivalentClasses>`
`  <Class IRI="Mother"/>`
`  <ObjectIntersectionOf>`
`    <Class IRI="Woman"/>`
`    <Class IRI="Parent"/>`
`  </ObjectIntersectionOf>`
`</EquivalentClasses>`
	`EquivalentClasses(`
	`  :Mother`
	`  ObjectIntersectionOf(`
	`    :Woman`
	`    :Parent`
	`  )`
	`)`

Extended OWL Example in Functional Syntax

- **Example 6.13.** The semantic network from 4.4 can be expressed in OWL (in functional syntax)



  - ClassAssertion formalizes the “inst” relation,
  - ObjectPropertyAssertion formalizes relations,
  - SubClassOf formalizes the “isa” relation,
  - for the “has_part” relation, we have to specify that *all birds have a part that is a wing* or equivalently *the class of birds is a subclass of all objects that have some wing*.

Extended OWL Example in Functional Syntax

- **Example 6.14.** The semantic network from 4.4 can be expressed in OWL (in functional syntax)

```
ClassAssertion (:Jack :robin)
ClassAssertion(:John :person)
ClassAssertion (:Mary :person)
ObjectPropertyAssertion(:loves :John :Mary)
ObjectPropertyAssertion(:owner :John :Jack)
SubClassOf(:robin :bird)
SubClassOf (:bird ObjectSomeValuesFrom(:hasPart :wing))
```

  - ClassAssertion formalizes the "inst" relation,
  - ObjectPropertyAssertion formalizes relations,
  - SubClassOf formalizes the "isa" relation,
  - for the "has_part" relation, we have to specify that *all birds have a part that is a wing* or equivalently *the class of birds is a subclass of all objects that have some wing*.

SPARQL an RDF Query language

- **Definition 6.15.** SPARQL, the "SPARQL Protocol and RDF Query Language" is an RDF query language, able to retrieve and manipulate data stored in RDF. The SPARQL language was standardized by the World Wide Web Consortium in 2008 [PS08].
- SPARQL is pronounced like the word "*sparkle*".
- **Definition 6.16.** A system is called a SPARQL endpoint, iff it answers SPARQL queries.
- **Example 6.17.** Query for person names and their e-mails from a triplestore with FOAF data.

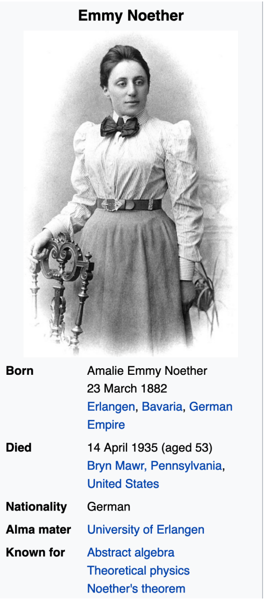
```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?email
WHERE {
  ?person a foaf:Person.
  ?person foaf:name ?name.
  ?person foaf:mbox ?email.
}
```

SPARQL Applications: DBPedia

- **Typical Application:** DBPedia screen-scrapes Wikipedia fact boxes for RDF triples and uses SPARQL for querying the induced triplestore.
- **Example 6.18 (DBPedia Query).** People who were born in Erlangen before 1900 (`http://dbpedia.org/snorql`)

```
SELECT ?name ?birth ?death ?person WHERE {
      ?person dbo:birthPlace :Erlangen .
      ?person dbo:birthDate ?birth .
      ?person foaf:name ?name .
      ?person dbo:deathDate ?death .
      FILTER (?birth < "1900−01−01"^^xsd:date) .
}
ORDER BY ?name
```

- The answers include Emmy Noether and Georg Simon Ohm.



A more complex DBPedia Query

- **Demo:** DBPedia `http://dbpedia.org/snorql/`
  Query: Soccer players born in a country with more than 10 M inhabitants, who play as goalie in a club that has a stadium with more than 30.000 seats.
  Answer: computed by DBPedia from a SPARQL query

```
SELECT distinct ?soccerplayer ?countryOfBirth ?team ?countryOfTeam ?stadiumcapacity
{
?soccerplayer a dbo:SoccerPlayer ;
   dbo:position|dbp:position <http://dbpedia.org/resource/Goalkeeper_(association_football)> ;
   dbo:birthPlace/dbo:country* ?countryOfBirth ;
   #dbo:number 13 ;
   dbo:team ?team .
   ?team dbo:capacity ?stadiumcapacity ; dbo:ground ?countryOfTeam .
   ?countryOfBirth a dbo:Country ; dbo:populationTotal ?population .
   ?countryOfTeam a dbo:Country .
FILTER (?countryOfTeam != ?countryOfBirth)
FILTER (?stadiumcapacity > 30000)
FILTER (?population > 10000000)
} order by ?soccerplayer
```

Results: Browse Go! Reset

SPARQL results:

soccerplayer	countryOfBirth	team	countryOfTeam	stadiumcapacity
:Abdesslam_Benabdellah	:Algeria	:Wydad_Casablanca	:Morocco	67000
:Airton_Moraes_Michellon	:Brazil	:FC_Red_Bull_Salzburg	:Austria	31000
:Alain_Gouaméné	:Ivory_Coast	:Raja_Casablanca	:Morocco	67000
:Allan_McGregor	:United_Kingdom	:Beşiktaş_J.K.	:Turkey	41903
:Anthony_Scribe	:France	:FC_Dinamo_Tbilisi	:Georgia_(country)	54549
:Brahim_Zaari	:Netherlands	:Raja_Casablanca	:Morocco	67000
:Bréiner_Castillo	:Colombia	:Deportivo_Táchira	:Venezuela	38755
:Carlos_Luis_Morales	:Ecuador	:Club_Atlético_Independiente	:Argentina	48069
:Carlos_Navarro_Montoya	:Colombia	:Club_Atlético_Independiente	:Argentina	48069
:Cristián_Muñoz	:Argentina	:Colo-Colo	:Chile	47000
:Daniel_Ferreyra	:Argentina	:FBC_Melgar	:Peru	60000
:David_Bičík	:Czech_Republic	:Karşıyaka_S.K.	:Turkey	51295
:David_Loria	:Kazakhstan	:Karşıyaka_S.K.	:Turkey	51295
:Denys_Boyko	:Ukraine	:Beşiktaş_J.K.	:Turkey	41903
:Eddie_Gustafsson	:United_States	:FC_Red_Bull_Salzburg	:Austria	31000

Triple Stores: the Semantic Web Databases

- **Definition 6.19.** A triplestore or RDF store is a purpose-built database for the storage RDF graphs and retrieval of RDF triples usually through variants of SPARQL.
- Common triplestores include
  - Virtuoso: `https://virtuoso.openlinksw.com/` (used in DBpedia)
  - GraphDB: `http://graphdb.ontotext.com/` (often used in WissKI)
  - blazegraph: `https://blazegraph.com/` (open source; used in WikiData)
- **Definition 6.20.** A description logic reasoner implements of reaonsing services based on a satisfiabiltiy test for description logics.
- Common description logic reasoners include
  - FACT++: `http://owl.man.ac.uk/factplusplus/`
  - HermiT: `http://www.hermit-reasoner.com/`
- **Intuition:** Triplestores concentrate on querying very large ABoxes with partial consideration of the TBox, while DL reasoners concentrate on the full set of ontology inference services, but fail on large ABoxes.

12.7 Ontologies vs. Databases

Example: Hogwarts Ontology

- **Example 7.1.** Axioms describe the structure of the world,

```
Class HogwartsStudent = Student and attendsSchool Hogwarts
Class: HogwartsStudent ⊑ hasPet only (Owl or Cat or Toad)
ObjectProperty: hasPet Inverses: isPetOf
Class: Phoenix ⊑ isPetOf only Wizard
```

- **Example 7.2.** Facts describe some particular concrete situation,

```
Individual: Hedwig
   Types: Owl
Individual: HarryPotter
  Types: HogwartsStudent
  Facts: hasPet Hedwig
Individual: Fawkes
  Types: Phoenix
  Facts: isPetOf Dumbledore
```

Ontologies vs. Databases

- **Obvious Analogy:** In an ontology:
  - axioms analogous to DB schema (structure and constraints on data)
  - facts analogous to DB data
    - data instantiates schema, is consistent with schema constraints
- **But there are also important differences:**

Database:

- Closed world assumption (CWA)
  - Missing information treated as false
- Unique name assumption (UNA)
  - Each individual has a single, unique name
- Schema behaves as constraints on structure of data
  - Define legal database states.

Ontology:

- Open world assumption (OWA)
  - Missing information treated as unknown
- No UNA
  - Individuals may have more than one name
- Ontology axioms behave like implications (inference rules)
  - Entail implicit information

DB vs. Ontology by Example (Querying)

- **Given the Ontology:**

  Individual: HarryPotter
    Facts: hasFriend RonWeasley
            hasFriend HermioneGranger
            hasPet Hedwig
  Individual: Draco Malfoy

- **Query:** Is Draco Malfoy a friend of HarryPotter?

DB vs. Ontology by Example (Querying)

- **Given the Ontology:**

```
Individual: HarryPotter
  Facts: hasFriend RonWeasley
         hasFriend HermioneGranger
         hasPet Hedwig
Individual: Draco Malfoy
```

- **Query:** Is Draco Malfoy a friend of HarryPotter?
  - DB: No
  - Ontology: Don't Know (OWA: didn't say Draco was not Harry's friend)

DB vs. Ontology by Example (Querying)

- **Given the Ontology:**

```
Individual: HarryPotter
  Facts: hasFriend RonWeasley
         hasFriend HermioneGranger
         hasPet Hedwig
Individual: Draco Malfoy
```

- **Query:** Is Draco Malfoy a friend of HarryPotter?
- **Counting Query:** How many friends does Harry Potter have?

DB vs. Ontology by Example (Querying)

- **Given the Ontology:**

```
Individual: HarryPotter
  Facts: hasFriend RonWeasley
         hasFriend HermioneGranger
         hasPet Hedwig
Individual: Draco Malfoy
```

- **Query:** Is Draco Malfoy a friend of HarryPotter?
- **Counting Query:** How many friends does Harry Potter have?
  - DB: 2
  - Ontology: at least 1 (No UNA: Ron and Hermione may be 2 names for same person)

DB vs. Ontology by Example (Querying)

- **Given the Ontology:**

```
Individual: HarryPotter
  Facts: hasFriend RonWeasley
         hasFriend HermioneGranger
         hasPet Hedwig
Individual: Draco Malfoy
```

- **Query:** Is Draco Malfoy a friend of HarryPotter?
- **Counting Query:** How many friends does Harry Potter have?
- **How about:** if we add

```
DifferentIndividuals: RonWeasley HermioneGranger
```

DB vs. Ontology by Example (Querying)

- **Given the Ontology:**

```
Individual: HarryPotter
  Facts: hasFriend RonWeasley
         hasFriend HermioneGranger
         hasPet Hedwig
Individual: Draco Malfoy
```

- **Query:** Is Draco Malfoy a friend of HarryPotter?
- **Counting Query:** How many friends does Harry Potter have?
- **How about:** if we add

```
DifferentIndividuals: RonWeasley HermioneGranger
```

  - DB: 2
  - Ontology: at least 2 (OWA: Harry may have more friends we didn't mention yet)

DB vs. Ontology by Example (Querying)

- **Given the Ontology:**

```
Individual: HarryPotter
  Facts: hasFriend RonWeasley
         hasFriend HermioneGranger
         hasPet Hedwig
Individual: Draco Malfoy
```

- **Query:** Is Draco Malfoy a friend of HarryPotter?
- **Counting Query:** How many friends does Harry Potter have?
- **How about:** if we add

```
DifferentIndividuals: RonWeasley HermioneGranger
```

- **And:** if we also add

```
Individual: HarryPotter
  Types: hasFriend only RonWeasley or HermioneGranger
```

DB vs. Ontology by Example (Querying)

- **Given the Ontology:**

```
Individual: HarryPotter
  Facts: hasFriend RonWeasley
         hasFriend HermioneGranger
         hasPet Hedwig
Individual: Draco Malfoy
```

- **Query:** Is Draco Malfoy a friend of HarryPotter?
- **Counting Query:** How many friends does Harry Potter have?
- **How about:** if we add

```
DifferentIndividuals: RonWeasley HermioneGranger
```

- **And:** if we also add

```
Individual: HarryPotter
  Types: hasFriend only RonWeasley or HermioneGranger
```

  - DB: 2
  - Ontology: 2

DB vs. Ontology by Example (Insertion)

- **Given:** the ontology from 7.1 and 7.2 insert

```
Individual: Dumbledore
Individual: Fawkes
  Types: Phoenix
  Facts: isPetOf Dumbledore
```

- **System Response:**

DB vs. Ontology by Example (Insertion)

- **Given:** the ontology from 7.1 and 7.2 insert

```
Individual: Dumbledore
Individual: Fawkes
  Types: Phoenix
  Facts: isPetOf Dumbledore
```

- **System Response:**
  - DB: Update rejected: constraint violation
    - Range of hasPet is Human; Dumbledore is not (CWA)
  - Ontology Reasoner:
    - Infer that Dumbledore is Human
    - Also infer that Dumbledore is a Wizard (only a Wizard can have a phoenix as a pet)

DB vs. Ontology by Example: Query Answering

- DB schema plays no role in query answering (efficiently implementable)
- Ontology axioms play a powerful and crucial role in QA
  - Answer may include implicitly derived facts
  - Can answer conceptual as well as extensional queries
    E.g., *Can a* Muggle *have a* Phoenix *for a* pet?
  - May have very high worst case complexity ($\widehat{=}$ terrible running time)
    Implementations may still behave well in typical cases.
- **Definition 7.3.** We call a query language semantic, iff query answering involves derived axioms and facts.
- **Observation 7.4.** *Ontology queries are semantic, while database queries are not.*

Summary: Ontology Based Information Systems

- Analogous to relational database management systems
  Ontology $\widehat{=}$ schema; instances $\widehat{=}$ data
- Some important (dis)advantages
  - + (Relatively) easy to maintain and update schema.
    - Schema plus data are integrated in a logical theory.
  - + Query results reflect both schema and data
  - + Can deal with incomplete information
  - + Able to answer both intensional and extensional queries
  - – Semantics may be counter-intuitive or even inappropriate
    - Open -vs- closed world; axioms -vs- constraints.
  - – Query answering much more difficult. (based on logical entailment)
    - Can lead to scalability problems.
- **In a nutshell** they deliver more valuable answers at cost of efficiency.

Chapter 13
The WissKI System: A Virtual Research Environment for Cultural Heritage

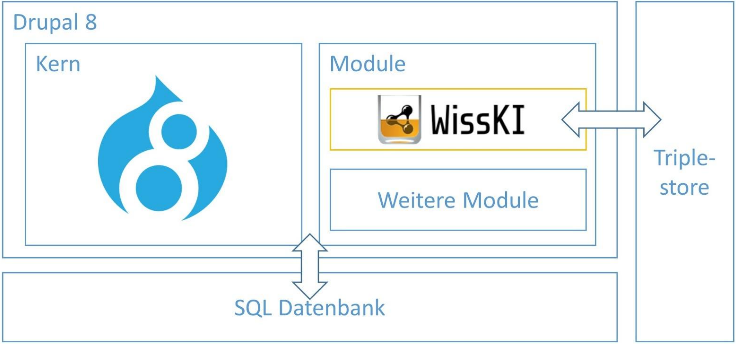
WissKI: a Virtual Research Env. for Cultural Heritage

- **Definition 0.1.** WissKI is a virtual research environment (VRE) for managing scholarly data and documenting cultural heritage.
- **Requirements:** For a virtual research environment for cultural heritage, we need
  - scientific communication about and documentation of the cultural heritage
  - networking knowledge from different disciplines (transdisciplinarity)
  - high-quality data acquisition and analysis
  - safeguarding authorship, authenticity, persistence
  - support of scientific publication
- WissKI was developed by the research group of Prof. Günther Görtz at FAU Erlangen-Nürnberg and is now used in hundreds of DH projects across Germany.
- FAU supports cultural heritage research by providing hosted WissKI instances.
  - See `https://wisski.data.fau.de` for details
  - We will use an instance for the Kirmes paintings in the homework assignments

13.1 WissKI extends Drupal

WissKI System Architecture

- Software basis: drupal CMS (content management system)
  - large, active community, extensible by drupal modules
  - provides much of the functionality of a VRE out of the box.



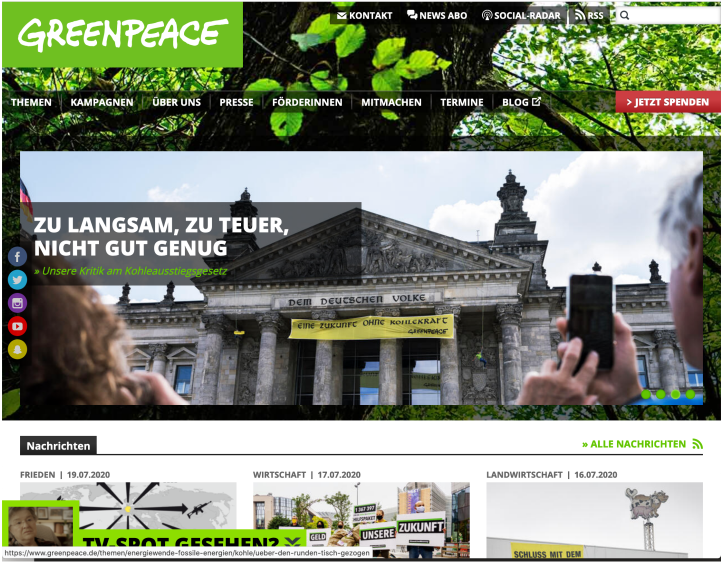
Drupal: A Web Content Managemt Framework

- **Definition 1.1.** Drupal is an open source web content management application. It combines CMS functionality with knowledge management via RDF.
- **Definition 1.2.** Drupal allows to configure web pages modularly from content blocks, which can be
  - static content, i.e. supplied by a module,
  - user supplied content, or
  - views, i.e. listings of content fragments from other blocks.

  These can be assembled into web pages via a visual interface: the config bar.

Assembling a Web Site via Drupal Blocks (Example)

- **Example 1.3 (Greenpeace via Drupal).** Can you find the blocks?



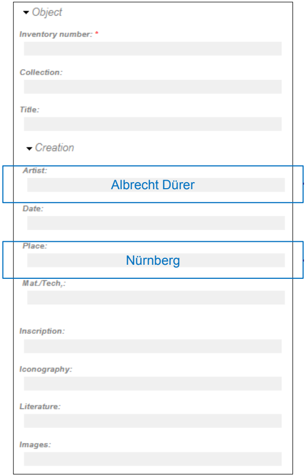
Drupal Modules and Themes

- **Idea:** Drupal is designed to be modular and extensible (so it can adapt to the ever-changing web)
- **Definition 1.4 (Modular Design).** Drupal functionality is structured into
  - drupal core – the basic CMS functionality
  - modules which contribute e.g. new block types ($\sim$ 45.000)
  - themes which contribute new UI layouts ($\sim$ 2800)

  Drupal core is the vanilla system as downloaded, modules and themes must be installed and configured separately via the config bar.
- The drupal core functionalities include
  - user/account management
  - menu management,
  - RSS feeds,
  - taxonomy,
  - page layout customization (via blocks and views),
  - system administration

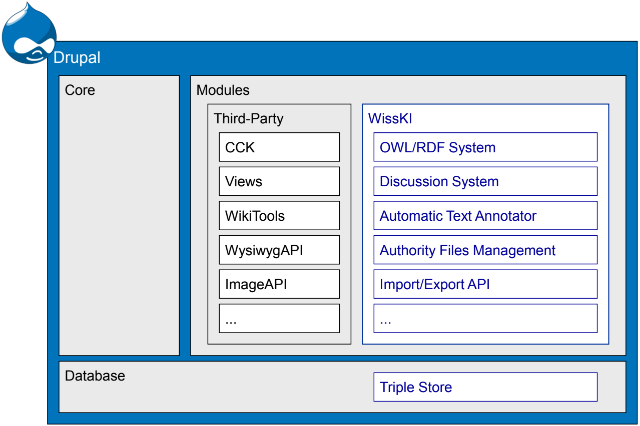
Bundles and Fields in Drupal (Data Entry)

- **Definition 1.5.** Drupal has a special data type called a bundle, which is essentially a dictionary: it contains key/value pairs called fields.
  - bundles can be nested $\leadsto$ sub bundles.
  - fields also have data type information, etc. to support editing.
- drupal presents bundles as
  - HTML lists for reading
  - HTML forms for data entry/editing
- Drupal bundles induce blocks that can be used for data entry and presentation.



WissKI System Architecture (Recap)

- WissKI = drupal + CIDOC CRM + triplestore + WissKI modules



- **Note:** Much of WissKI functionality is configurable via the drupal config bar.

13.2 Dealing with Ontology Paths: The WissKI Pathbuilder

The WissKI Path Builder (Idea)

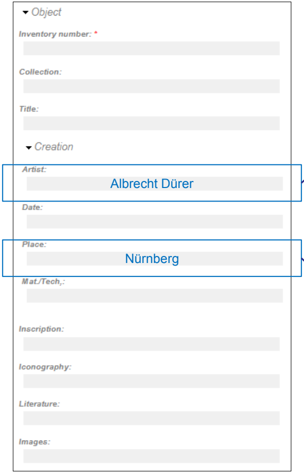
- **Recall:** *Albrecht Dürer painted Melencolia 1 in Nürnberg*

$$\begin{tikzcd}
 & & d : E21 \arrow[r, "P131"] & a : E82 \arrow[r, "P3"] & \texttt{"A. Dürer"} \\
m : E84 \arrow[r, "P108i"] & q : E12 \arrow[ur, "P14"] \arrow[dr, "P7"] & & & \\
 & & p : E53 \arrow[r, "P87"] & n : E48 \arrow[r, "P3"] & \texttt{"Nürnberg"}
\end{tikzcd}$$

- **Idea:** Hide the complexity induced by the ontology from the user
  - Form-based interaction with categories and fields (as in a RDBMS UI)
- **Definition 2.1.** The WissKI path builder maps ontology groups and ontology paths to drupal bundles and fields.
  - ontology groups become data entry forms (bundles) for the root entities,
  - their fields are mapped to ontology paths.
  - subtrees in the ontology become sub-bundles. (shared objects)

The WissKI Path Builder (Example)

- **Example 2.2 (A WissKI Group).**

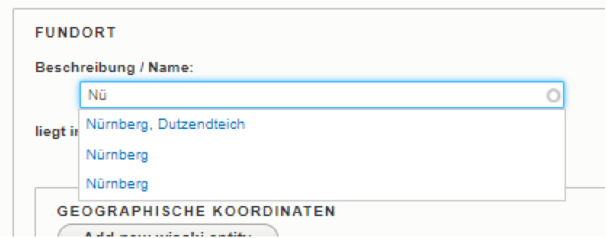


E84 Information Carrier
→ P108i was produced by →
E12 Production
→ P14 carried out by →
E21 Person
→ P131 is identified by →
E82 Actor Appellation
→ P3 has note →
„Albrecht Dürer“

E84 Information Carrier
→ P108i was produced by →
E12 Production
→ P7 took place at →
E53 Place
→ P87 is identified by →
E48 Place Name
→ P3 has note →
„Nürnberg “

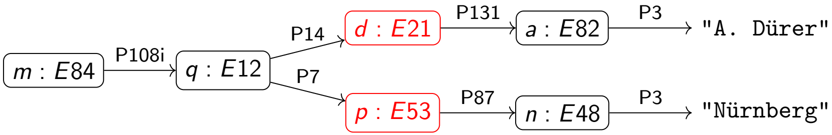
Sharing and Disambiguation in Path Builders

- **Observation 2.3.** *Sometimes we want to refer to existing entities in WissKI.*
- **Example 2.4 (Referring to Nürnberg).** (We love tab completion)



Sharing and Disambiguation in Path Builders

- **Observation 2.8.** *Sometimes we want to refer to existing entities in WissKI.*
- **Example 2.9 (Referring to Nürnberg).** (We love tab completion)
- **Example 2.10 (To What).** Albrecht Dürer created all his etchings in Nürnberg.
- **Problem:** (In paths) we are creating lots of objects, which ones to offer?
- **Idea:** Mark the entities we might want to reuse on paths while specifying them.
- **Definition 2.11.** A disambiguation point in a path marks an entity that can be re used in data acquisition.
- **Example 2.12.** Disambiguation points are highlighted in red on paths.



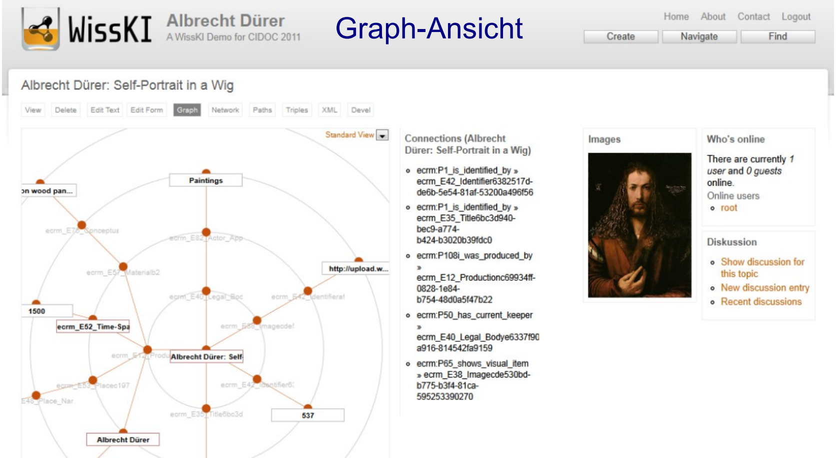
Specifying/Maintaining WissKI Path Builders

- **Recall:** A WissKI path builder maps ontology groups and ontology paths to drupal bundles and fields.
- **Example 2.13 (Specifying a WissKI Path Builder).**

TITLE	PATH	ENABLED	FIELD TYPE	CARDINALITY	OPERATIONS
Werk	Group [ecrm:E22_Man-Made_Object]	✓		Unlimited	Edit
Titel	ecrm:E22_Man-Made_Object -> ecrm:P102_has_title -> ecrm:E35_Title	✓	Text (plain)	1	Edit
Verwalter	ecrm:E22_Man-Made_Object -> ecrm:P50_has_current_keeper -> ecrm:E40_Legal_Body -> ecrm:P1_is_identified_by -> ecrm:E82_Actor_Appellation	✓	Text (plain)	1	Edit
Inventarnummer	ecrm:E22_Man-Made_Object -> ecrm:P1_is_identified_by -> ecrm:E42_Identifier	✓	Text (plain)	1	Edit
Beziehung	ecrm:E22_Man-Made_Object -> ecrm:P46i_forms_part_of -> ecrm:E22_Man-Made_Object -> ecrm:P102_has_title -> ecrm:E35_Title	✓	Text (plain)	Unlimited	Edit
Herstellung	Group [ecrm:E22_Man-Made_Object -> ecrm:P108i_was_produced_by -> ecrm:E12_Production]	✓		Unlimited	Edit
Hersteller	ecrm:E22_Man-Made_Object -> ecrm:P108i_was_produced_by -> ecrm:E12_Production -> ecrm:P14_carried_out_by -> ecrm:E21_Person -> ecrm:P131_is_identified_by -> ecrm:E82_Actor_Appellation	✓	Text (plain)	Unlimited	Edit
Datum	ecrm:E22_Man-Made_Object -> ecrm:P108i_was_produced_by -> ecrm:E12_Production -> ecrm:P4_has_time-span -> ecrm:E52_Time-Span	✓	Text (plain)	1	Edit
Ort	ecrm:E22_Man-Made_Object -> ecrm:P108i_was_produced_by -> ecrm:E12_Production -> ecrm:P7_took_place_at -> ecrm:E53_Place -> ecrm:P1_is_identified_by -> ecrm:E44_Place_Appellation	✓	Text (plain)	Unlimited	Edit
Material	ecrm:E22_Man-Made_Object -> ecrm:P108i_was_produced_by -> ecrm:E12_Production -> ecrm:P32_used_general_technique -> ecrm:E57_Material -> ecrm:P1_is_identified_by -> ecrm:E75_Conceptual_Object_Appellation	✓	Text (plain)	Unlimited	Edit
Technik	ecrm:E22_Man-Made_Object -> ecrm:P108i_was_produced_by -> ecrm:E12_Production -> ecrm:P33_used_specific_technique -> ecrm:E29_Design_or_Procedure -> ecrm:P1_is_identified_by -> ecrm:E75_Conceptual_Object_Appellation	✓	Text (plain)	Unlimited	Edit
Kommentar	ecrm:E22_Man-Made_Object -> ecrm:P129i_is_subject_of -> ecrm:E31_Document	✓	Text (formatted, long)	1	Edit
Abbildung	ecrm:E22_Man-Made_Object -> ecrm:P138i_has_representation -> ecrm:E36_Visual_Item -> ecrm:P1_is_identified_by -> ecrm:E51_Contact_Point	✓	Image	Unlimited	Edit

WissKI Path Builders as Graphs

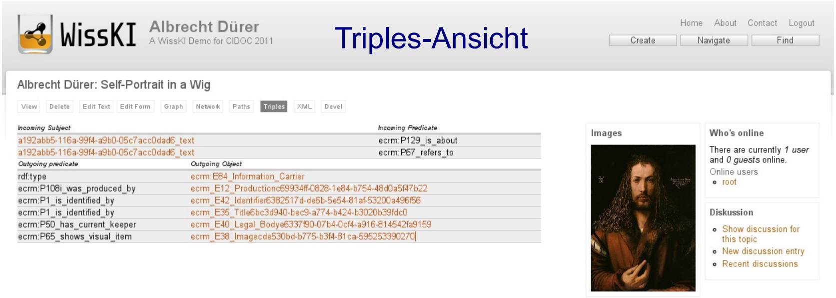
- **Example 2.14 (A WissKI Path Construtor as a Graph).**



- Very nice and helpful, but does not work currently!

WissKI Path Builders as Triples

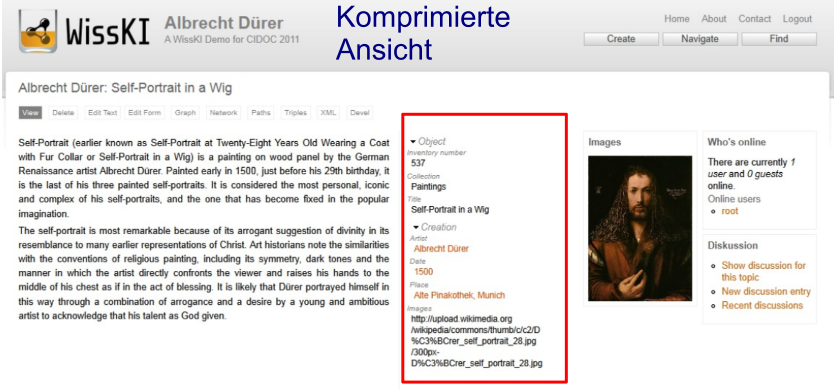
- **Of course** we can view path builders as sets of triples.
- **Example 2.15 (A WissKI Path Construtor as Triples).**



- Such an export also allows standardized communication.

Data Presentation using Path Builders in WissKI

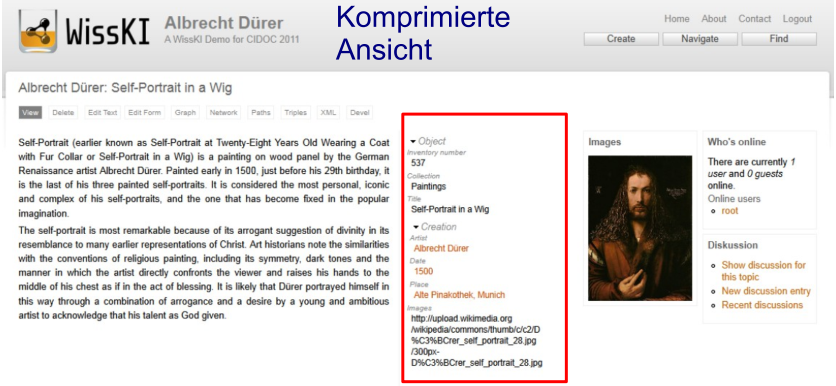
- Path builders can be used as drupal blocks for data presentation.
  - For every object o, aggregate the values of the paths starting in o.
- **Example 2.16 (Compressed View).**



13.3 The WissKI Link Block

The WissKI Link Block (Idea)

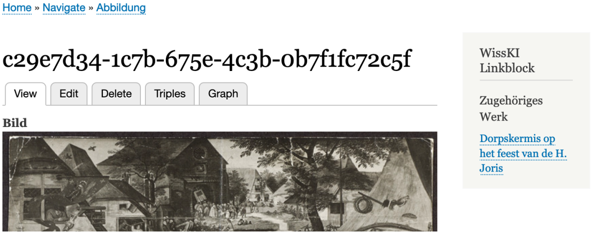
- **Observation 3.1.** *For an entity in a RDF graph, both the outgoing and the incoming relations are important for understanding.*
- **Example 3.2.** This view only shows the outgoing edges!



- **Idea:** Add a block with "incoming links" to the page, use the path builder.

Link Blocks (Definition)

- **Definition 3.3.** Let p be a drupal page for an ontology group g, then a WissKI link block is a special drupal block with associated path builder, whose ontology paths all end in g.
- **Example 3.4 (A link block for Images).**



Note the difference between

- a "work" – the original painting Pieter Brueghel created in 1628
- and an "image of the work" – a b/w photograph of the "work".

This particular link block mediates between these two.

A Link Block in the Wild (the full Picture)

- **Example 3.5 (A link block for Images).**

Home » Navigate » Abbildung

c29e7d34-1c7b-675e-4c3b-0b7f1fc72c5f

View | Edit | Delete | Triples | Graph

Bild

Bild-URL
http://kirmes.wisski.agfd.fau.de/sites/default/files/2020-07/c29e7d34-1c7b-675e-4c3b-0b7f1fc72c5f_.jpg
Bild-ID
c29e7d34-1c7b-675e-4c3b-0b7f1fc72c5f
Lizenz
CC BY-NC-SA 4.0
Kommentar
Es handelt sich um den Scan einer s/w-Fotografie. Die Fotografie weist einige Knicke an den Ecken sowie kleinere Risse auf.

WissKI Linkblock

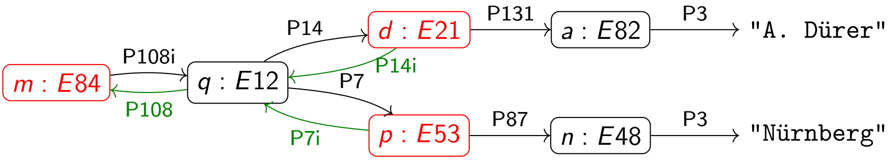
Zugehöriges Werk

Dorpskermis op het feest van de H. Joris

- **outgoing relations** below the image,
- **incoming ones** in the link block

Making Link Blocks via the Path Builder

- How to make a link block in page p for group g? (Details at [WH])
  1. create a block via the config bar and place it on p.
  2. associate it with a link block path builder
  3. model paths into g in the path builder (various source groups)
- **Idea:** You essentially know link block paths already: If you have already modeled a path $g, r_1, \ldots, r_n, s$ for a group s, then you have a path $s, r_n^{-1}, \ldots, r_1^{-1}, g$, where r_i^{-1} are the inverse roles of r_i (exist in CIDOC CRM)



- **Note:** With this setup, you never have to fill out the link block paths!

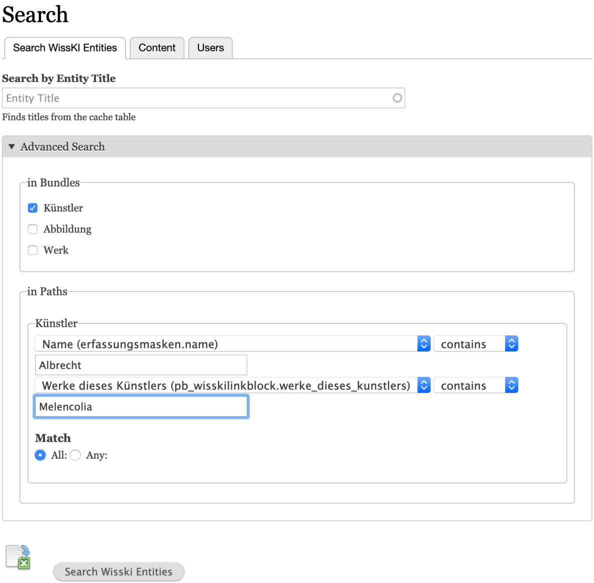
13.4 Cultural Heritage Research: Querying WissKI Resources

Research in WissKI

- **So far** we have seen how to acquire complex knowledge about cultural artefacts using CIDOC CRM ABoxes.
- **Question:** But how do we do research using WissKI?
- **Answer:** Finding patterns, inherent connections, ...in the data.
- **But how?:** That depends on the kind of research you want to do. Here are some WissKI research tools
  1. we can use drupal search on the data.
  2. We can formulate our own queries in SPARQL
  3. We can pre-configure various queries in drupal views.

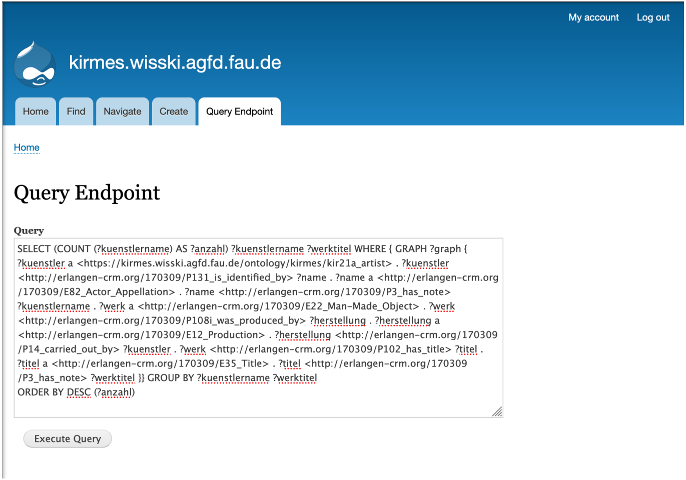
Drupal Search in WissKI

- **Example 4.1.**



SPARQL Endpoint in WissKI

- **Example 4.2.** Find kirmes paintings and their painters and count them



SPARQL Endpoint in WissKI

- **Example 4.3.** Find kirmes paintings and their painters and count them

kirmes.wisski.agfd.fau.de

Home | Find | Navigate | Create | Query Endpoint

Home

Query Endpoint

?anzahl	?kuenstlername	?werktitel
"2"^^xsd:integer	"Pieter Brueghel (II)"	" Dorpskermis op het feest van de H. Joris "
"1"^^xsd:integer	"Pieter Brueghel (II)"	"Dorpskermis op het feest van de H. Joris"

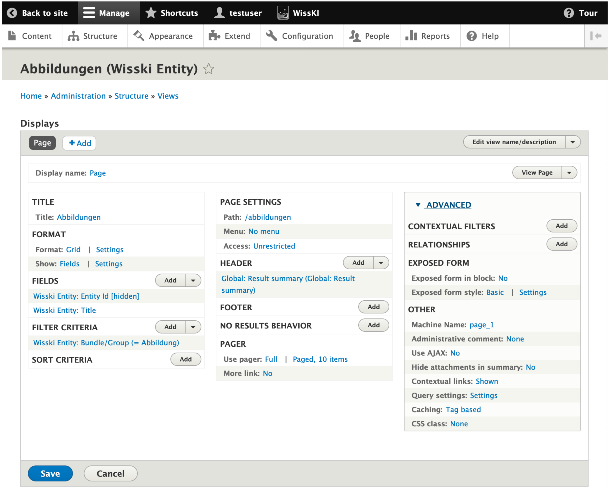
Query

```
SELECT (COUNT (?kuenstlername) AS ?anzahl) ?kuenstlername ?werktitel WHERE { GRAPH ?graph {
?kuenstler a <https://kirmes.wisski.agfd.fau.de/ontology/kirmes/kir21a_artist> . ?kuenstler
<http://erlangen-crm.org/170309/P131_is_identified_by> ?name . ?name a <http://erlangen-crm.org
/170309/E82_Actor_Appellation> . ?name <http://erlangen-crm.org/170309/P3_has_note>
?kuenstlername . ?werk a <http://erlangen-crm.org/170309/E22_Man-Made_Object> . ?werk
```

Execute Query

Data Presentation via Views in WissKI

- **Example 4.4 (Configuring a View).** This makes a drupal block.



Drupal generates a SPARQL query, aggregates results into a block.

This Research is WissKI-instance-local

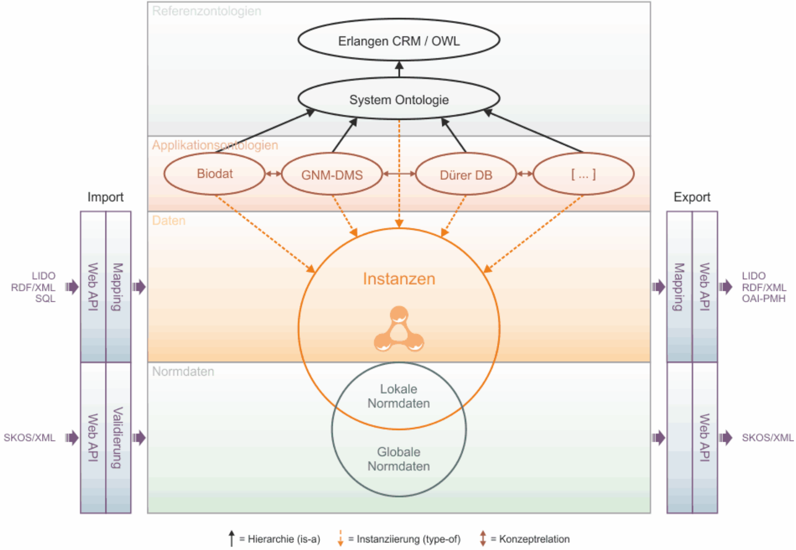
- **Observation 4.5.** *All these research queries only work in the current WissKI instance.*
- **Observation 4.6.** *There is probably much more about the entities you are interested in outside your particular WissKI instance.*
- **Problem:** How to make use of this?
- **Solution:** We need to do two things
  1. Make use of other people's ABoxes
  2. Provide your ABox to other people.

  This practice is called linked open data. (up next)

13.5 Application Ontologies in WissKI

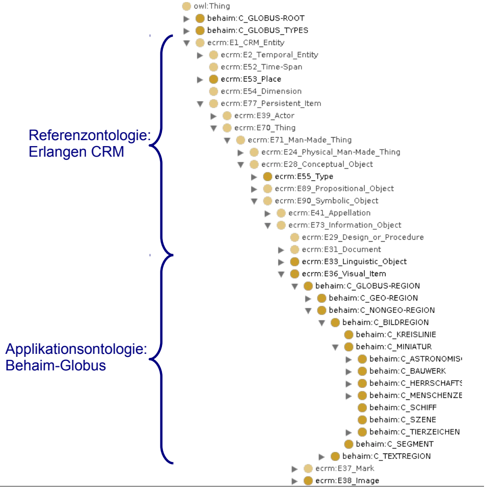
WissKI Information Architecture (Ontologies)

- Ontologies, instances, and export formats



Application Ontologies extend CIDOC CRM

- **Observation 5.1.** *Sometimes we need more than CIDOC CRM.*
- **Definition 5.2.** A WissKI application ontology is one that extends CIDOC CRM, without changing it.
- **Example 5.3 (Behaim Application Ontology).**



Making an Application Ontology

- The “current ontology” of a WissKI instance can be configured via the config bar via the “WissKI ontology” module.
- The application ontology should import CIDOC CRM.
- **Idea:** Use Protg for that.

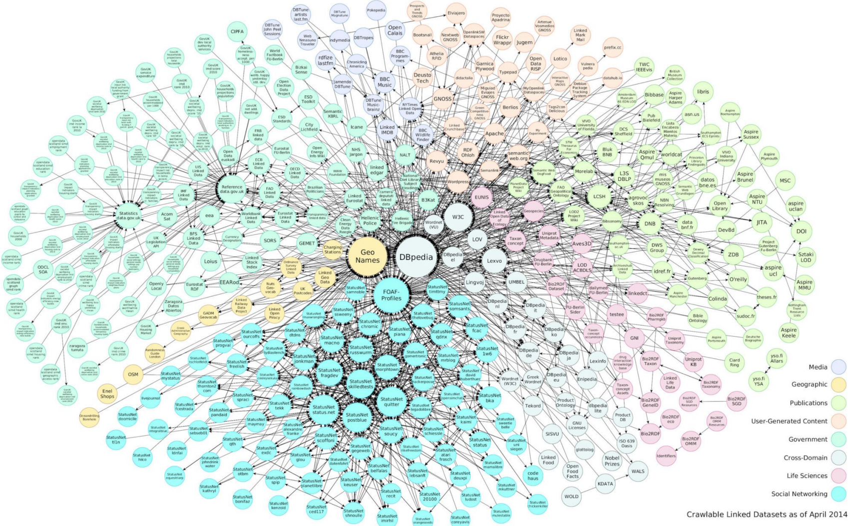
13.6 The Linked Open Data Cloud

Linked Open Data

- **Definition 6.1.** Linked data is structured data in which classified objects are interlinked via relations with other objects so that the data becomes more useful through semantic queries and access methods.
- **Definition 6.2.** Linked open data (LOD) is linked data which is released under an open license, which does not impede its reuse by the community.
- **Definition 6.3.** Given the semantic web technology stack, we can create interoperable ontologies and interlinked data sets, we call their totality the .
- **Recall the LOD Incentives:**
  - incentivize other authors to extend/improve the LOD
    $\rightsquigarrow$ more/better data can be generated at a lower cost.
  - generate *attention* to the LOD and recognition for authors
    $\rightsquigarrow$ this gives alternative revenue models for authors.

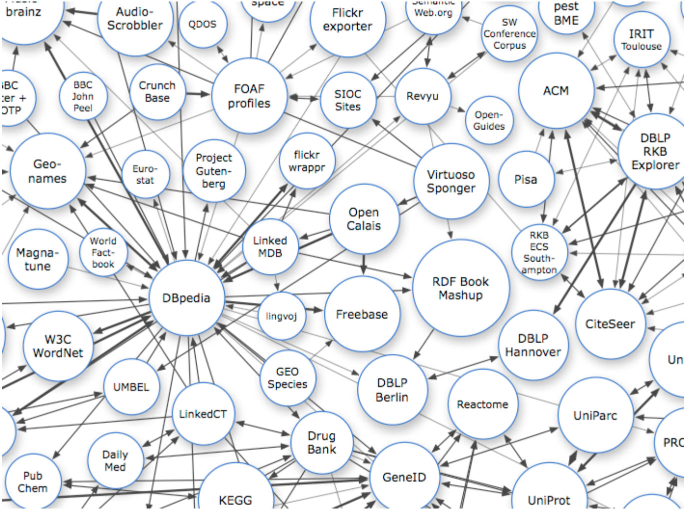
The Linked Open Data Cloud

- The linked open data cloud in 2014 (today much bigger, but unreadable)



The Linked Open Data Cloud

- Zooming in (data sets and their – interlinked – ontologies)



Using the LOD-Cloud in WissKI

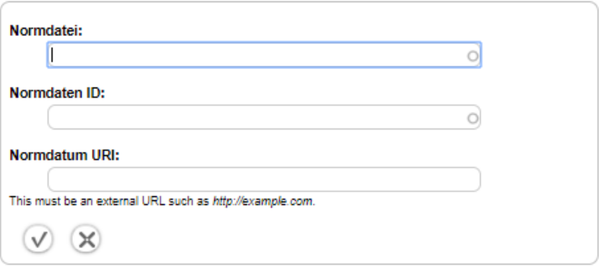
- **Idea:** Do not re-model entities that already exist (in the LOD Cloud)
- **Problem:** Most of the LOD Cloud is about things we do not want.
- But there are some sources that are useful
  - the GND (Gemeinsame Normdatei [GND]), an authority file for personal/corporate names and keywords from literary catalogs,
  - geonames[GN], a geographical database with more than 25M names and locations
  - Wikipedia
- **Observation 6.4.** *All of them provide URIs for real world entities, which is just what we need for objects in RDF triples.*
- **Definition 6.5.** WissKI provides special modules called adapters for GND and geonames.

Using Geonames in WissKI (Example)

1. **Example 6.6.** We want to use the "Meilwald" (Erlangen) in WissKI.

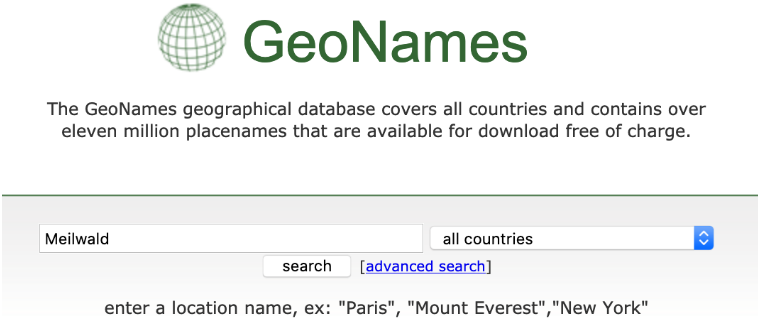
Using Geonames in WissKI (Example)

1. **Example 6.7.** We want to use the "Meilwald" (Erlangen) in WissKI.
2. make a sub-ontology groups "norm data" in the WissKI path builder
3. The induced sub-bundle looks like this:



Using Geonames in WissKI (Example)

1. **Example 6.8.** We want to use the "Meilwald" (Erlangen) in WissKI.
2. make a sub-ontology groups "norm data" in the WissKI path builder
3. The induced sub-bundle looks like this:
4. We enter `https://geodata.org` for "Normdatei" and go there to find out the URI for "Meilwald" which goes into "Normdatum URI".



Using Geonames in WissKI (Example)

1. **Example 6.9.** We want to use the "Meilwald" (Erlangen) in WissKI.
2. make a sub-ontology groups "norm data" in the WissKI path builder
3. The induced sub-bundle looks like this:
4. We enter `https://geodata.org` for "Normdatei" and go there to find out the URI for "Meilwald" which goes into "Normdatum URI".
5. there may be multiple results (here only one)

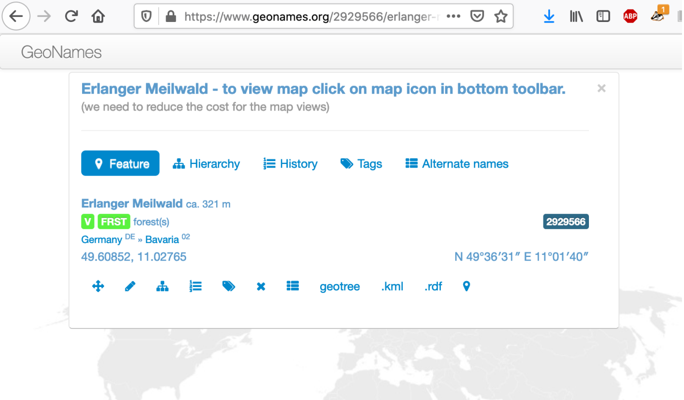

Meilwald | all countries
search [advanced search]

1 records found for "Meilwald"

	Name	Country	Feature class	Latitude	Longitude
1	Erlanger Meilwald Erlanger Meil-Wald,Erlanger Meilwald,Meilwald	Germany, Bavaria	forest(s)	N 49° 36' 30''	E 11° 1' 39''

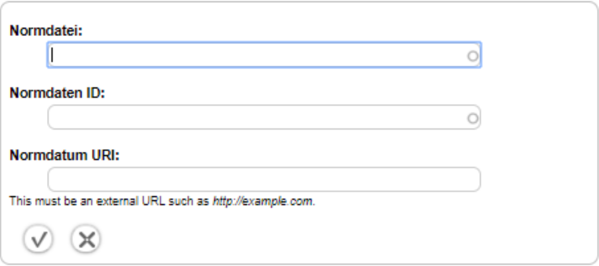
Using Geonames in WissKI (Example)

1. **Example 6.10.** We want to use the "Meilwald" (Erlangen) in WissKI.
2. make a sub-ontology groups "norm data" in the WissKI path builder
3. The induced sub-bundle looks like this:
4. We enter `https://geodata.org` for "Normdatei" and go there to find out the URI for "Meilwald" which goes into "Normdatum URI".
5. there may be multiple results (here only one)
6. Select/click the intended one, check the details



Using Geonames in WissKI (Example)

1. **Example 6.11.** We want to use the "Meilwald" (Erlangen) in WissKI.
2. make a sub-ontology groups "norm data" in the WissKI path builder
3. The induced sub-bundle looks like this:
4. We enter `https://geodata.org` for "Normdatei" and go there to find out the URI for "Meilwald" which goes into "Normdatum URI".
5. there may be multiple results (here only one)
6. Select/click the intended one, check the details
7. Enter the URL from the URL bar into "Normdatum URI".



Towards a WissKI Commons in the LOD Cloud

- **Recap:** We can directly refer to (URIs of) external objects in WissKI.
- **Observation 6.12.** *The most interesting source for references to cultural artefacts are other WissKI instances.*
- **Problem:** A WissKI is an island, unless it exports its data! (few do)
- **Idea:** We need a LOD cloud of cultural heritage research data under to foster object centric research in the humanities.
- **Definition 6.13.** We call the part of this resource that can be created by aggregating WissKI exports the WissKI commons.
- **Observation 6.14.** *WissKI exports meet the FAIR principles quite nicely already.*
- We will be working on a FAU WissKI commons in the next years. (help wanted)

References I

[CC] *CIDOC CRM - The CIDOC Conceptual Reference Model*. url: http://www.cidoc-crm.org/ (visited on 07/13/2020).

[CQ69] Allan M. Collins and M. Ross Quillian. "Retrieval time from semantic memory". In: *Journal of verbal learning and verbal behavior* 8.2 (1969), pp. 240–247. doi: 10.1016/S0022-5371(69)80069-1.

[ECRMa] *erlangen-crm*. url: https://github.com/erlangen-crm (visited on 07/13/2020).

[ECRMb] *Erlangen CRM/OWL - An OWL DL 1.0 implementation of the CIDOC Conceptual Reference Model (CIDOC CRM)*. url: http://erlangen-crm.org/ (visited on 07/13/2020).

[FAIR18] European Commission Expert Group on FAIR Data. *Turning FAIR into reality*. 2018. doi: 10.2777/1524.

[FOAF14] *FOAF Vocabulary Specification 0.99*. Namespace Document. The FOAF Project, Jan. 14, 2014. url: http://xmlns.com/foaf/spec/.

[GN] *Geonames*. url: https://www.geonames.org/ (visited on 07/29/2020).

References II

[GND] *DNB – The Integrated Authority File (GND)*. url: https://www.dnb.de/EN/Professionell/Standardisierung/GND/gnd_node.html (visited on 07/29/2020).

[Her+13] Ivan Herman et al. *RDFa 1.1 Primer – Second Edition. Rich Structured Data Markup for Web Documents*. W3C Working Goup Note. World Wide Web Consortium (W3C), Apr. 19, 2013. url: http://www.w3.org/TR/xhtml-rdfa-primer/.

[JS] *json – JSON encoder and decoder*. url: https://docs.python.org/3/library/json.html (visited on 04/16/2021).

[KC04] Graham Klyne and Jeremy J. Carroll. *Resource Description Framework (RDF): Concepts and Abstract Syntax*. W3C Recommendation. World Wide Web Consortium (W3C), Feb. 10, 2004. url: http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/.

[LM] *LabelMe: the open annotation tool*. url: http://labelme.csail.mit.edu (visited on 08/28/2020).

References III

[LXML] *lxml – XML and HTML with Python*. url: https://lxml.de (visited on 12/09/2019).

[Nor+18a] Emily Nordmann et al. *Lecture capture: Practical recommendations for students and lecturers*. 2018. url: https://osf.io/huydx/download.

[Nor+18b] Emily Nordmann et al. *Vorlesungsaufzeichnungen nutzen: Eine Anleitung für Studierende*. 2018. url: https://osf.io/e6r7a/download.

[OWL09] OWL Working Group. *OWL 2 Web Ontology Language: Document Overview*. W3C Recommendation. World Wide Web Consortium (W3C), Oct. 27, 2009. url: http://www.w3.org/TR/2009/REC-owl2-overview-20091027/.

[Pro] *Protégé*. Project Home page at http://protege.stanford.edu. url: http://protege.stanford.edu.

[PRR97] G. Probst, St. Raub, and Kai Romhardt. *Wissen managen*. 4 (2003). Gabler Verlag, 1997.

References IV

[PS08] Eric Prud'hommeaux and Andy Seaborne. *SPARQL Query Language for RDF*. W3C Recommendation. World Wide Web Consortium (W3C), Jan. 15, 2008. url: http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/.

[SUMO] *Suggested Upper Merged Ontology*. url: http://www.adampease.org/OP/ (visited on 01/25/2019).

[UL] *urllib – URL handling modules*. url: https://docs.python.org/3/library/urllib.html (visited on 04/15/2021).

[WH] *WissKI Handbuch*. url: http://wiss-ki.eu/documentation/wisski_handbuch (visited on 07/23/2020).

[Wil+16] Mark D. Wilkinson et al. "The FAIR Guiding Principles for scientific data management and stewardship". In: *Scientific Data* 3 (2016). doi: 10.1038/sdata.2016.18.