XML, JSON & NoSQL Databases

Gilles Degols
based on the initial work of Ken Hasselmann

Course organization

- XML, JSON
  - Theory and exercises
- NoSQL Databases
  - Theory
  - Developing a small python application iteratively through exercises
- Not everything will be written in the slides
  - If you do not come to the class, you will miss some information needed for the evaluations
  - Slides can be updated at any time, as well as the project/exercises deliverables (communicated orally)
  - https://github.com/gilles-degols/ecam-nosql
- Deliverables: must be in English

Evaluation - Exercises

- XML, JSON (10%)
  - Submit exercises the next day of each related course (23:59)
- NoSQL Databases (20%)
  - Submit exercises the next day of each related course (23:59)
- Submit: g3d@ecam.be with "Exercises: {XML/JSON/NoSQL}" as title
  - "{lastname} {firstname}.zip"
    - Only .zip
    - No sharepoint
  - Late submission: 0/20 to the related evaluation
- Exercises are *personal*
  - You should still help each other
  - 0/20 for copy-paste
  - Trying to hide by changing variable names is a bad idea

Evaluation - Project

- Project (70%)
  - 3-people teams unless exception
  - Design, implementation & setup of the database in a docker-compose.yml + application
  - Presentation (.pdf) and code (app + database setup) must be sent ***the day before*** the evaluation at 23:59 the latest
  - Last course: 20 minutes presentation + 20 minutes Q/A
    - Time allocation is free to change if deemed necessary by the lecturer
  - Everyone will listen to every presentation
  - Different notes can be given depending on the contribution & comprehension of each student
- Submit: g3d@ecam.be with "Project - Team {i}" + .zip
  - Late submission: 0/20 to the related evaluation

Evaluation - Project

Database & Implementation justification	Feature implementation	Rating mark
Yes	Yes (full scope)	[14; 20]
Yes	No (full scope not done)	[0; 14[
No	Yes	0
No	No	[0; 14[

Sending the code is part of "feature implementation"
(no code, no feature)

About the lecturer

- Software Engineer / Big Data → Data Engineer
- Teaching Assistant at Université Libre de Bruxelles
- Companies I worked for
  - Université Libre de Bruxelles
  - Macq
  - ADB Safegate
  - Evonik
  - Proximus
  - Engie GEMS
- Course content
  - Directly related to day-to-day work

Intro to XML

Why?
How to use it ?
How to validate it ?

XML: Why ?

- Extensible Markup Language
- Markup language
- File format

- Goals
  - Simplicity
  - Generality
  - Usability
- To communicate data in a structured format (!= HTML)

XML: Why ?

- SGML: Standard Generalized Markup Language
  - Released in 1986
  - Enable sharing of machine-readable documents, for several decades
- HTML is a variant of SGML
  - Pre-defined tags
  - Presentation layer
- XML is a variant of SGML
  - Data layer

XML: Why ?

- Define your logging (log4j)

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration debug="true" xmlns:log4j='http://jakarta.apache.org/log4j/'>
   <appender name="console" class="org.apache.log4j.ConsoleAppender">
      <layout class="org.apache.log4j.PatternLayout">
           <param name="ConversionPattern" value="%d{yyyy-MM-dd HH:mm:ss}" />
      </layout>
   </appender>
   <root>
      <level value="DEBUG" />
      <appender-ref ref="console" />
   </root>
</log4j:configuration>
```

XML: Why ?

- Define your build settings (maven)

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1</version>
  <dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-api</artifactId>
    <version>1.2.17</version>
  </dependency>
</project>
```

- And a multitude of other use cases across a lot of applications, languages, build tools, …,
- Also extensively used to transfer data

XML: Why ?

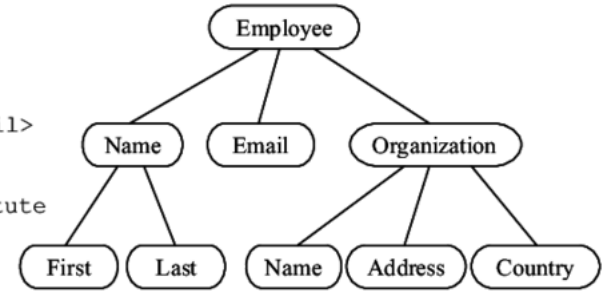
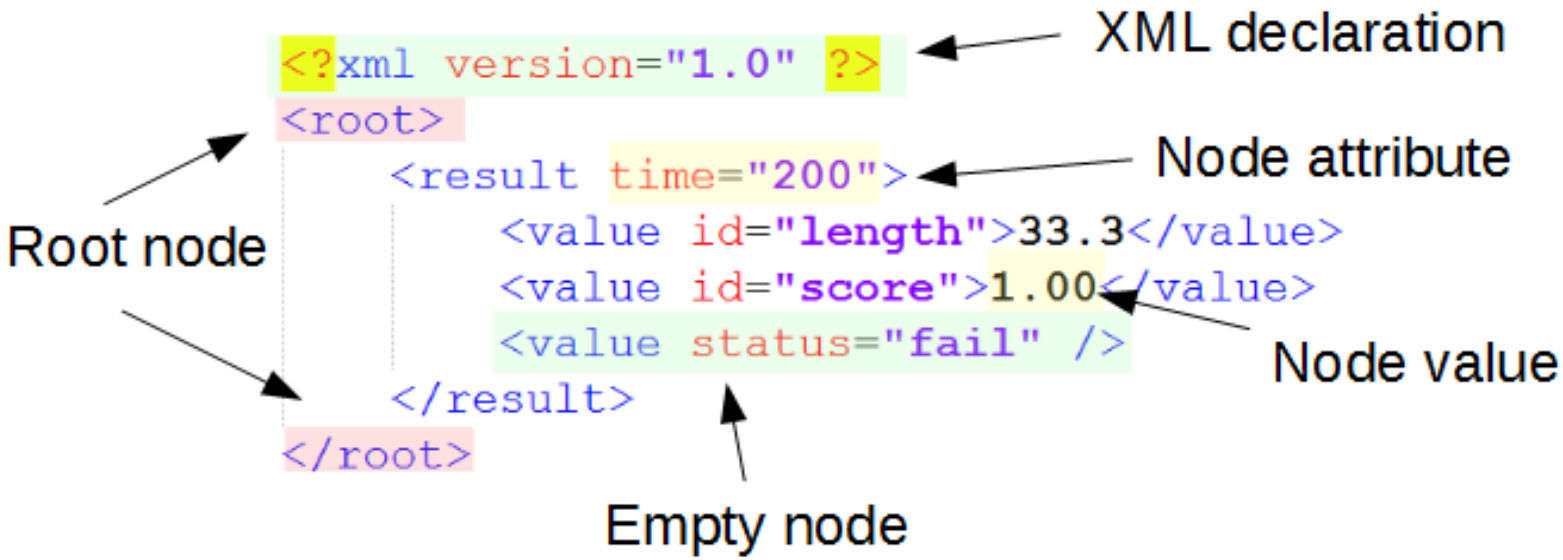
- Send & Receive data from an API (SOAP)
  - Envelope: identifies the XML document as SOAP message
  - Header: header information (authentication, …)
  - Body: call & response
  - Fault: errors & status

XML: Why ?

- SOAP
  -
```
<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

  <soap:Header>
  ...
  </soap:Header>

  <soap:Body>
  ...
    <soap:Fault>
    ...
    </soap:Fault>
  </soap:Body>

</soap:Envelope>
```

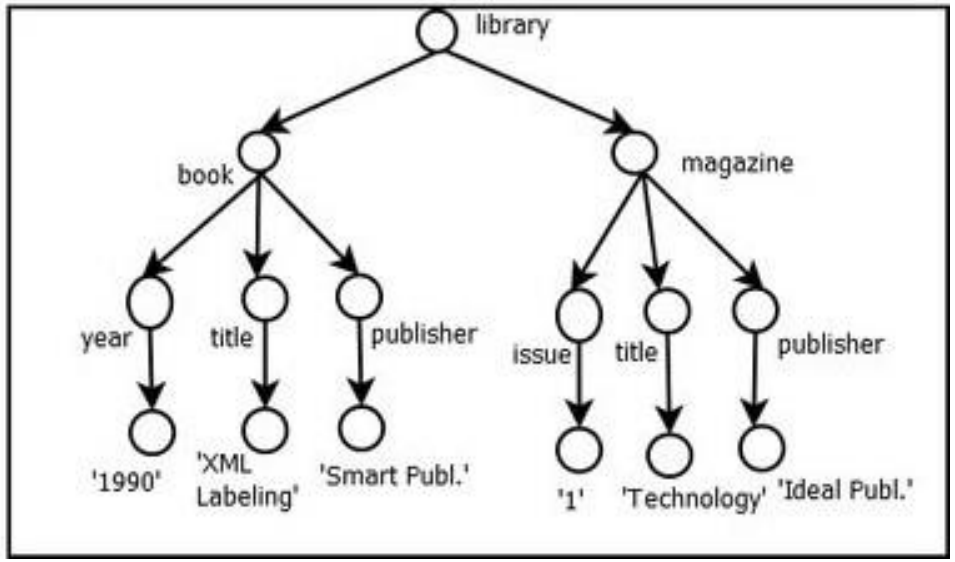
XML: Why ?

- SVG – Scalable Vector Graphics
  - ```
    <svg height="100" width="100" xmlns="http://www.w3.org/2000/svg">
      <circle r="45" cx="50" cy="50" stroke="green" stroke-width="3"
        fill="red" opacity="0.5" />
    </svg>
    ```


    ```

XML: How to use it ?

- Standard XML syntax, v1.0 (5th edition)
  - Released in 1998
  - Public format: https://w3.org/TR/xml
- Most languages have an XML library
- Structure definition (and validation)
  - DTD
  - XML Schema (XSD)

XML: Some properties

- An XML document is *well-formed* if it follows the syntax rules of XML
- An XML document is *valid* if it is *well-formed* and follows the structured defined in a Document Type Definition (DTD) or XML Schema (XSD)
- An XML document does not contain any information on how it should be rendered

XML: Declaration

- `<?xml version="1.0" encoding="UTF-8" ?>`
- Basic information for the XML parser:
  - XML version
  - Character encoding (optional) - most of the time, UTF-8
- But, how would you read the encoding of the first line without knowing the encoding ?

XML: Structure

```
<Employee>
    <Name>
        <First>Lassi</First>
        <Last>Lehto</Last>
    </Name>
    <Email>Lassi.Lehto@fgi.fi</Email>
    <Organization>
        <Name>
            Finnish Geodetic Institute
        </Name>
        <Address>
            PO Box 15,
            FIN-02431 Masala
        </Address>
        <Country CountryCode="358">Finland</Country>
    </Organization>
</Employee>
```



- Root: `Employee`
- Nodes: `Name, Email, …`
- Attributes: `CountryCode=358`

XML: Structure




```
<?xml version="1.0" ?>
<root>
    <result time="200">
        <value id="length">33.3</value>
        <value id="score">1.00</value>
        <value status="fail" />
    </result>
</root>
```

XML: Structure

- All elements start with a start tag and end with an end tag
- The name of the element is formed using
  - Alphanumeric characters a-zA-Z0-9
  - Underscore, dash, dot
  - Colons (:) are possible but they define a namespace
  - No space
  - Does not start with a number
  - Does not start with "xml"

XML: Namespace

- Within an XML Schema, you might want to re-use some tags
- Namespaces
  - `log4j:configuration`
  - `soap:body`
- You must define them
  - For html code: `xmlns="http://www.w3.org/TR/html4/"`
  - `xmlns:log4j="http://jakarta.apache.org/log4j/ "`
  - `xmlns:soap="http://www.w3.org/2003/05/soap-envelope"`
- Default namespace
  - Avoid always putting the namespace as prefix

XML: Elements

- Start and end tag must correspond
- No crossings: `<intro>…<title>…</intro>…</title>`
- Case sensitive: `<Title>` and `<title>` are different tags
- Only one root element
  - At the top of the document
  - Cannot appear again elsewhere in the tree
- XML comments: `<!-- comment -->`

XML: Elements

- Elements can be:
  - Non empty: start with opening tag and end with closing tag, can contain text and other elements
    - `<title>The lord of the rings<title>`
  - Empty: do not contain text nor other elements
    - `<title></title>` or `<title />`
- Elements can have attributes:
  - `<title type="fantasy">The lord of the rings<title>`
  - Attributes should be defined between quotes (') or double quotes (")

XML: Elements

```
<parent>
  <sibling1> ... </sibling1>
  <sibling2> ... </sibling2>
  <self>
    <child1>
      ... <desc1></desc1> ... <desc2></desc2> ...
    </child1>
    <child2> ... </child2>
    <child3>
      ... <desc3><desc4> ... </desc4></desc3> ...
    </child3>
  </self>
  <sibling3> ... </sibling3>
</parent>
```

XML: Related features

- XLink
  - Link to other xml documents, like "href" in html
- XPath
  - `/bookstore/book[1]/title`
  - `/bookstore/book[price>35]/price`
- XQuery
  - The SQL for your XMLs
  - ```
    for $x in doc("books.xml")/bookstore/book
    where $x/price>30
    return $x/title
    ```
- …
    ```

XML - Exercise 1

- Transform the following XML tree into a valid XML file (by hand)



XML - Exercise 2

- Transform the following recipe you received from a friend into an XML file (by hand)
  - The XML is going to be used by a website to show all ingredients (wherever they appear), total execution time, necessary tooling …
  - Make sure the generated xml is consistent and easy to process by a software
- Recipe for Japanese Curry
- Ingredients
  - Beef, chopped: 450g
  - Onions, minced: 350g
  - Carrot, chopped: 100g
  - Potato, chopped: 150g
  - Water: 500ml
  - Golden Curry Sauce Mix: 92g
- Directions
  - Stir-fry meat and vegetables with oil in a large skillet on medium heat for approx. 5 min.
  - Add water and bring to boil. Reduce heat, cover and simmer until ingredients are tender, approx. 15min.
  - Turn the heat off, break S&B Golden Curry Sauce Mix into pieces and add them to the skillet. Stir until sauce mixes are completely melted. Simmer approx. 5 min., stirring constantly.
  - Serve hot over rice or noodles.

DTD: What is it?

- Defines structural constraints in XML
- The Document Type Definition (DTD) defines the elements and their rules
- A document - with a related DTD - is valid if:
  - It is well-formed
  - It references a DTD
  - It complies with the DTD

External DTD

- The DTD can be included directly in the document, or in an external file
- External DTD
  - `<!DOCTYPE root_element SYSTEM|PUBLIC [name] DTD_uri>`
  - `<!DOCTYPE people_list SYSTEM "example.dtd">`
    `<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"`
    `"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">`
- The DOCTYPE allows to declare the type of the document, the identifier for the root element is needed
  - SYSTEM : is local to computer, PUBLIC: can be retrieved from a catalog

Internal DTD

- The DTD can be directly included in the document file
  - <!DOCTYPE people_list [
    ...
    ]>

DTD: Example

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE people_list [
  <!ELEMENT people_list (person*)>
  <!ELEMENT person (name, birthdate?, gender?,
socialsecuritynumber?)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT birthdate (#PCDATA)>
  <!ELEMENT gender (#PCDATA)>
  <!ELEMENT socialsecuritynumber (#PCDATA)>
]>
<people_list>
  <person>
    <name>Fred Bloggs</name>
    <birthdate>2008-11-27</birthdate>
    <gender>Male</gender>
  </person>
</people_list>
```

DTD: Issues

- A DTD can be used to create a denial-of-service attack by defining nested entities expanding exponentially, or by sending the XML parser to an external resource that never returns
- Many frameworks & software (Microsoft Office) will not open files containing DTD declarations
- Other issues
  - It does not use an XML syntax
  - No typing of content
  - No regex matching
- → Replaced by XML Schema

XML Schema: Overview

- Describe the structure of an XML document
- XML Document

```
<?xml version="1.0"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

XML Schema: Overview

- DTD Rules

```
<!ELEMENT note (to, from, heading, body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

XML Schema: Overview

- XSD

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="https://www.w3schools.com"
xmlns="https://www.w3schools.com"
elementFormDefault="qualified">

<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="heading" type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
```

XML Schema - Benefits

- Introduce data types
- Use XML
  - Same language
  - Same parser
  - Same editor
- Extensible
  - Re-use a Schema in other Schemas
  - Create your own data type
  - Use multiple schemas in the same document

XML Schema: Another example

- XML

```
<?xml version="1.0"?>
<Book>
    <Title>XML Schema Essentials</Title>
    <Authors>
        <Author>R. Allen Wyke</Author>
        <Author>Andrew Watt</Author>
    </Authors>
    <Publisher>John Wiley</Publisher>
</Book>
```

XML Schema: Another example

- XSD

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:annotation>
        <xsd:documentation>
            This is a sample XML Schema for Chapter 1 of XML Schema
            Essentials.
        </xsd:documentation>
    </xsd:annotation>
    <xsd:element name="Book">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="Title"/>
                <xsd:element ref="Authors"/>
                <xsd:element ref="Publisher"/>
            </xsd:sequence>
            <xsd:attribute name="pubCountry" type="xsd:string"/>
        </xsd:complexType>
    </xsd:element>
```

…

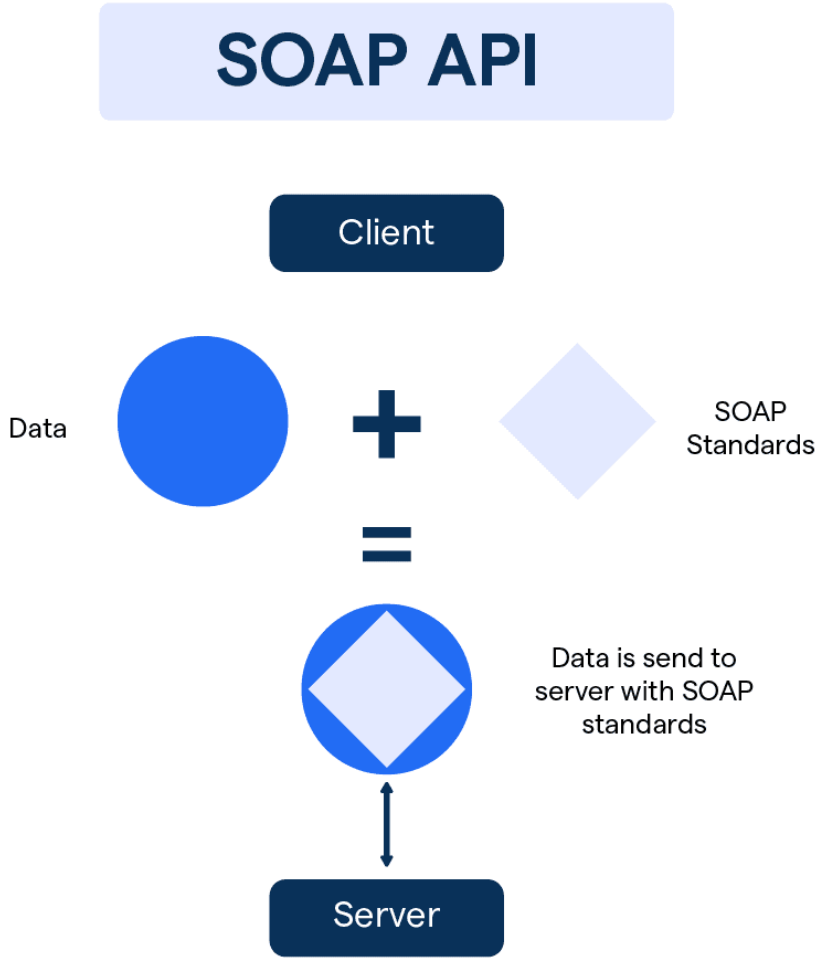
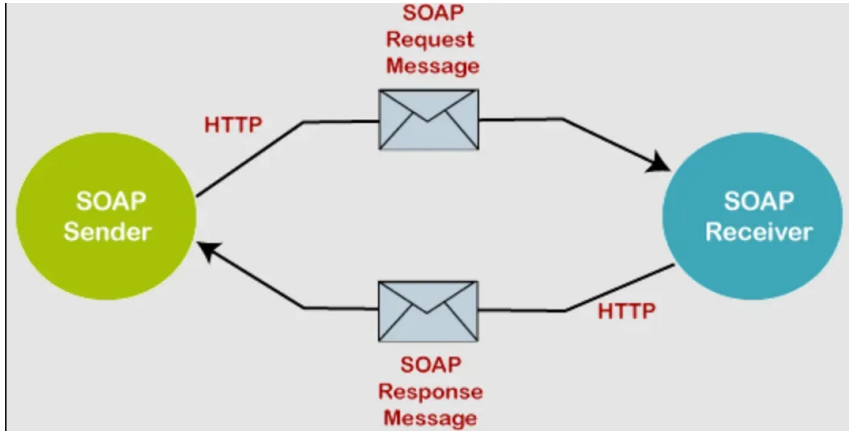
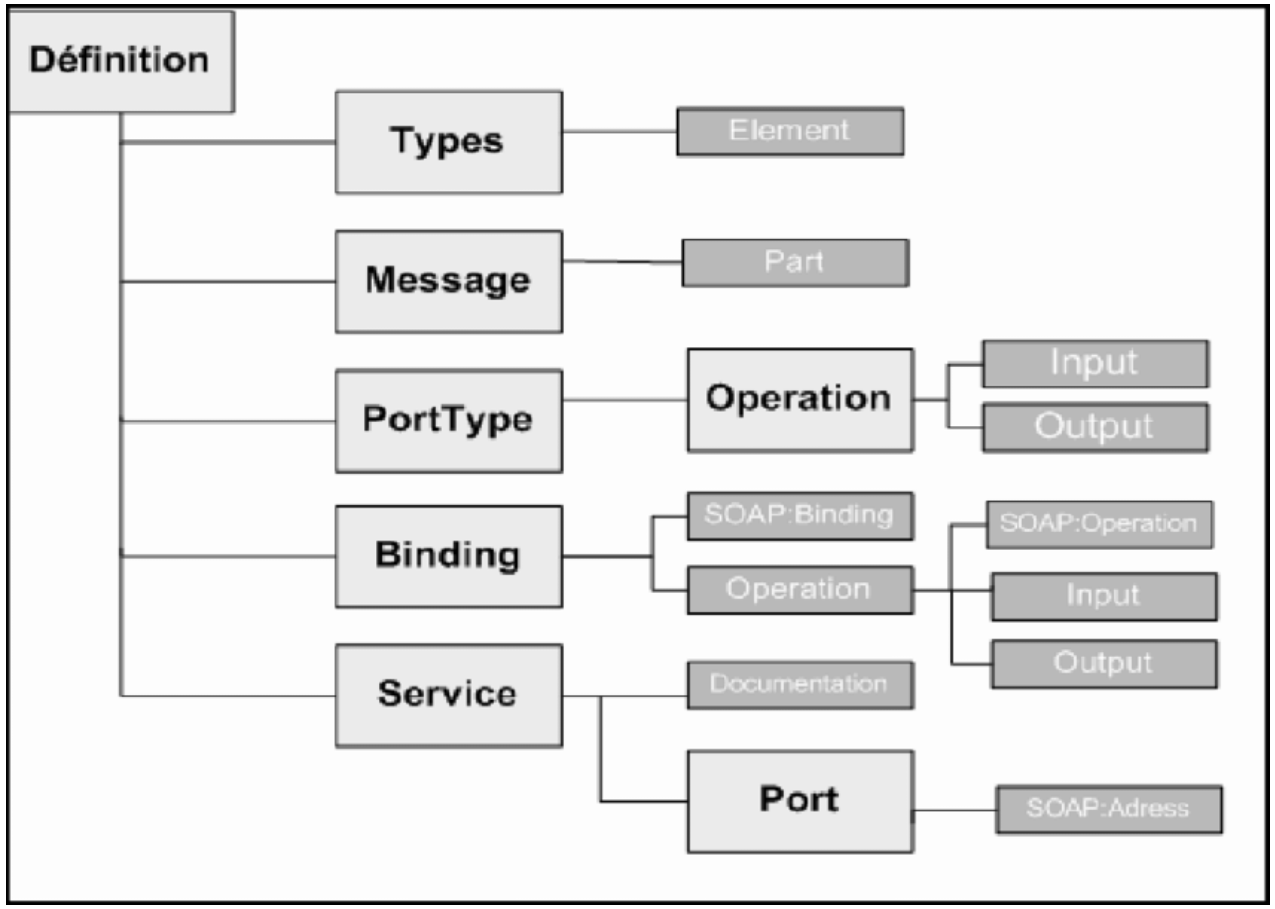
XML Schema: Another example

```
…
    <xsd:element name="Title" type="xsd:string"/>
    <xsd:element name="Authors">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="Author" minOccurs="1"
maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="Author" type="xsd:string"/>
    <xsd:element name="Publisher" type="xsd:string"/>
</xsd:schema>
```

XML Schema - A few keywords

- Tags
  - `element`
  - `complexType`
  - `sequence`
  - `attribute`
- Attributes
  - `type`
  - `name`
  - `maxOccurs`
  - `minOccurs`
  - `ref`

XML - Exercise 3

- Create a XSD to validate the following XML
- ```
  <?xml version="1.0" encoding="UTF-8"?>
  <Race date="2020-12-15" name="Holiday Meet">
      <Course>
          <CourseName>The track</CourseName>
          <Address>Track road 123</Address>
      </Course>
      <Horses>
          <Horse name="Bonfire">
              <Value>5000</Value>
              <Birthdate>1998-05-01</Birthdate>
              <Gender>M</Gender>
          </Horse>
          <Horse name="Dobby">
              <Value>1000</Value>
              <Birthdate>2001-04-05</Birthdate>
              <Gender>F</Gender>
          </Horse>
      </Horses>
  </Race>
  ```
  ```

XML - Exercise 4

- Use Python and the library lxml to load the xml of any given file, and list the content of any given xpath
- Example:
  - python ex4.py myfile.xml /Race/Horses/Horse
  - `b'<Horse name="Bonfire">\n\t\t\t<Value>5000</Value>\n\t\t\t<Birthdate>1998-05-01</Birthdate>\n\t\t\t<Gender>M</Gender>\n\t\t</Horse>\n\t\t\n'`
  - `b'<Horse name="Dobby">\n\t\t\t<Value>1000</Value>\n\t\t\t<Birthdate>2001-04-05</Birthdate>\n\t\t\t<Gender>F</Gender>\n\t\t</Horse>\n\t\n'`

XML & HTTP: SOAP



SOAP

- Simple Object Access Protocol
- Enveloppe
  - Root element with XML namespaces
- Header
  - Optional
  - Authentication tokens, encryption details, custom headers, …
- Body
  - Payload itself
- Fault
  - Error codes, error messages



SOAP - Request

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="https://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <m:GetUser>
      <m:UserId>123</m:UserId>
    </m:GetUser>
  </soap:Body>
</soap:Envelope>
```

SOAP - Response

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="https://www.w3.org/2003/05/soap-envelope/"
               soap:encodingStyle="https://www.w3.org/2003/05/soap-
encoding">
    <soap:Body>
        <m:GetUserResponse>
            <m:Username>Tony Stark</m:Username>
        </m:GetUserResponse>
    </soap:Body>
</soap:Envelope>
```

SOAP API

- XSD
  - Describe the structure of the data types being exchanged
  - Describe the fields and restrictions on fields (max length, …)
- WSDL - Web Services Description Language
  - Describe the API and its operations
  - List of methods, parameters and returned values
  - Abstract definitions of endpoints and messages separated from the network deployment or data format bindings

SOAP API - WSDL

- Definitions
  - `targetNamespace`
  - `xmlns`: default namespace of the WSDL document
  - `xmlns:tns` current namespace
- Types
  - Contains various xsd
- Message
- Operation
- portType
- Binding
- Port
- service

SOAP API - WSDL



SOAP API

- Demo

XML - Exercise 5

- Use Python to create a SOAP API providing the various features
  - In memory "database"
  - (Shop) objects
    - Attributes: name, remaining quantity, price
    - List, create, update & delete
  - Orders
    - Attributes: object_id, customer_id, quantity
    - List, create, update & delete
  - Apply some basic validations
    - quantity >= 0, name must be of length [4;100], birth date is a real date ...
- Provide a python script to test each endpoint

XML - Credits & references

- Ken Hasselmann
  - Introduction au XML: https://brunomartin.be/cours/xml.pdf
  - Working with XML trees: https://docs.fab-image.com/studio/programming_tips/UsingXml.html
  - XML documentation: https://www.w3.org/XML/
- Official XML Schema tutorial from w3schools
  - https://www.w3schools.com/xml/schema_intro.asp
- XML Schema Essentials
  - https://nuleren.be/ebooks/xml-schema-essentials.pdf
- Japanese curry recipe
  - https://www.sbfoods-worldwide.com/recipes/010.html

Intro to JSON

Why?
How to use it ?
How to validate it ?

JSON: What ?

- **J**ava**S**cript **O**bject **N**otation
- Text format to store and transport data
- Self-describing and easy to read
- ```
  {
    "property": "value",
    "hello \" $'": null,
    "some-key": [
      {
        "id": 42,
        "is_valid": true,
        "precision": 42.23
      }
    ]
  }
  ```
  ```

JSON: Syntax

- Syntactically similar to creating JavaScript objects
  - JSON.parse(), JSON.stringify()
- Syntactically similar to creating Python objects
  - json.loads(), json.dumps()
- Syntax rules
  - Data is in key/value pairs
  - Data is separated by commas (careful about an extra ",")
  - Curly braces hold objects
  - Square brackets hold arrays

JSON: Syntax

- One single root object
- Supported data types
  - String
  - Number
  - Object
  - Array
  - Boolean
  - Null
- `null`: a valid json (case sensitive!)
  - But also: `"some-string", 40.0, {}, [], true`

JSON: Accessing data

- Python
  - ```
    import json
    data = json.loads("""{"property": "value","some-key": [{"id":
    42,"is_valid": true}]}""")
    print(data["property"])
    print(data["some-key"][0]["id"])
    ```
  - ```
    data = {"property": "value", "some-key": [{"id": 42, "is_valid":
    True}]}
    print(json.dumps(data))
    ```

XML vs JSON

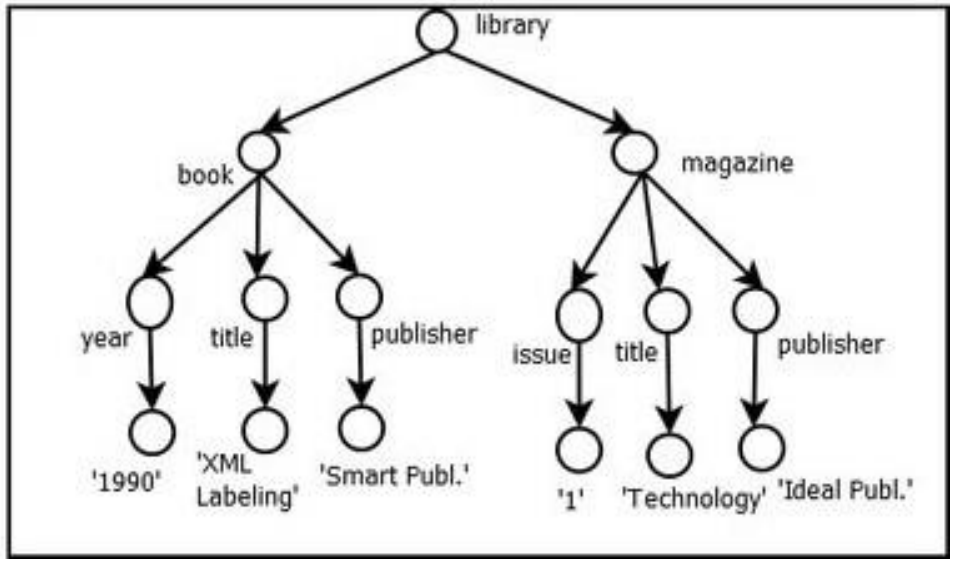
XML	JSON
Human readable	
Hierarchical	
Supported by most languages	
Legacy solution, wildly supported	Wildly supported
Specific parser needs to be implemented	Quick to read & write
Does not translate directly into basic python/javascript structures	Fast to parse
	Smaller data storage
	No start/end tags

JSON: Applications

- SVG: still XML
- Config files (npm packaging, vscode configuration …)
- HTTP: SOAP API → (JSON) API
- Websocket
  - `{"id":42, "value": 56.0, "symbol": "BTCUSD"}`
  - …
- Databases
  - Structured format, so why not store it directly this way?
  - NoSQL Databases (and some SQL databases)

JSON - Exercise 1

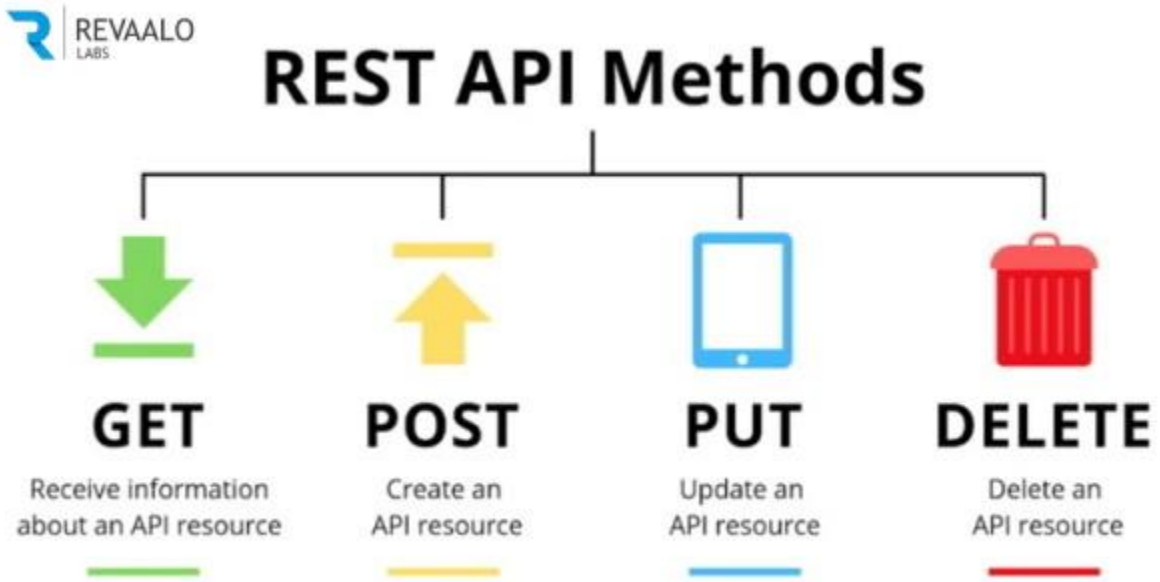
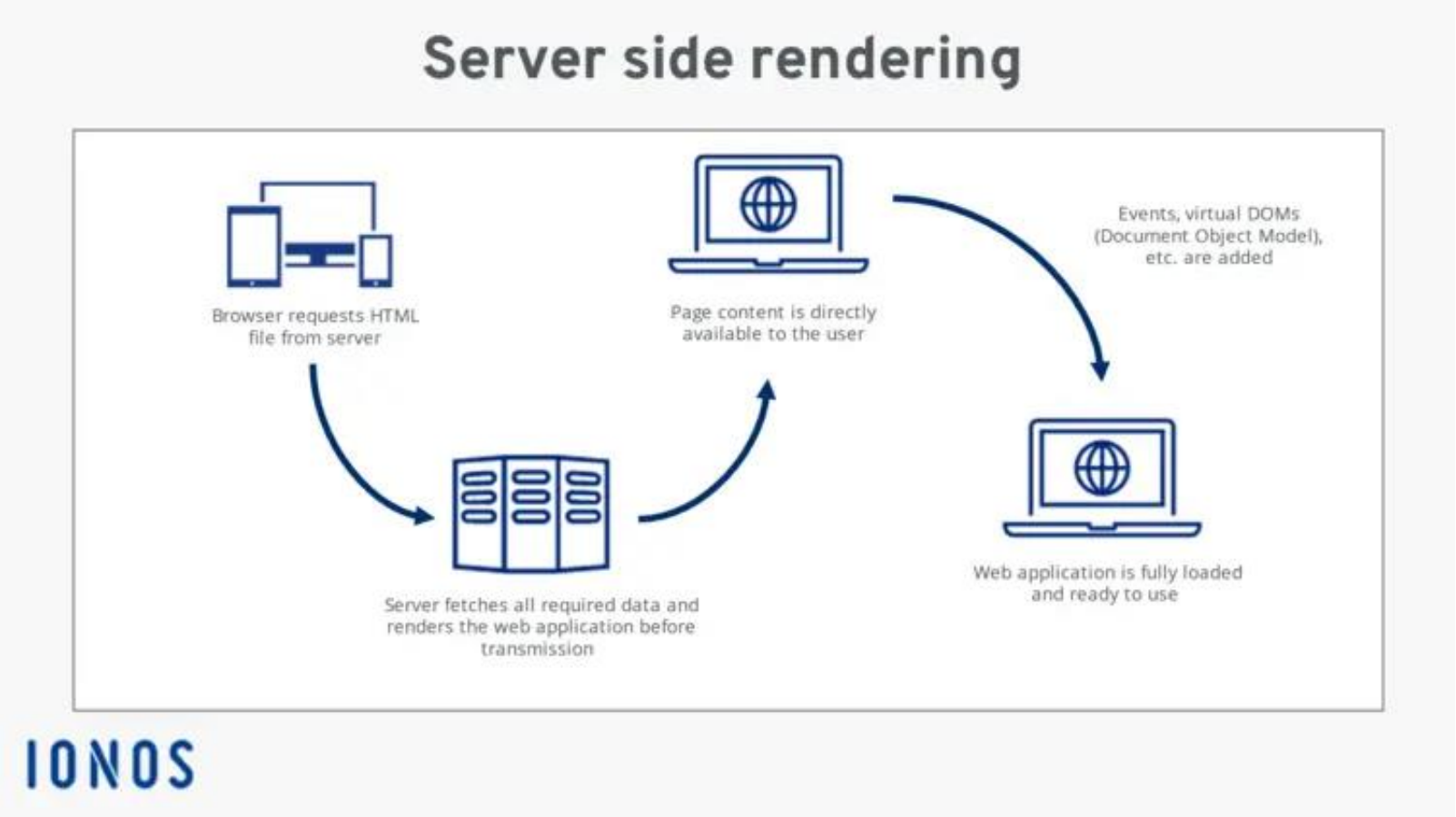
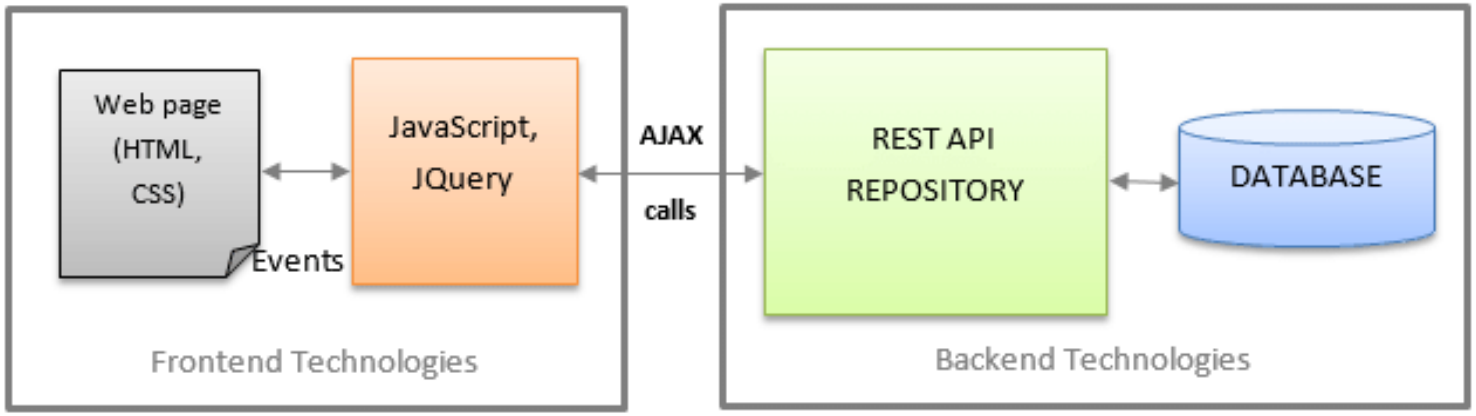
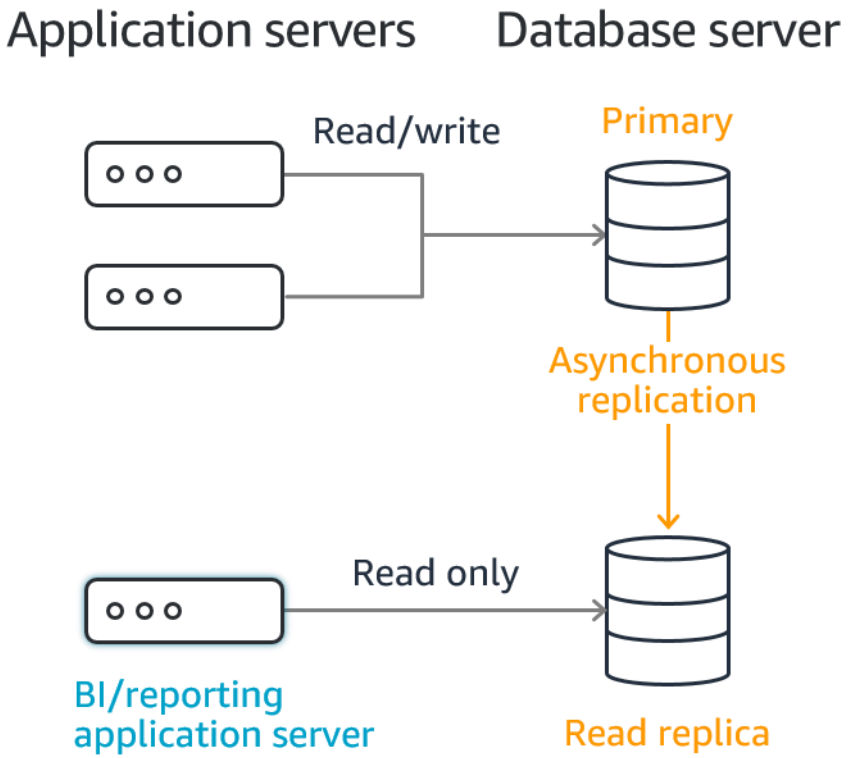
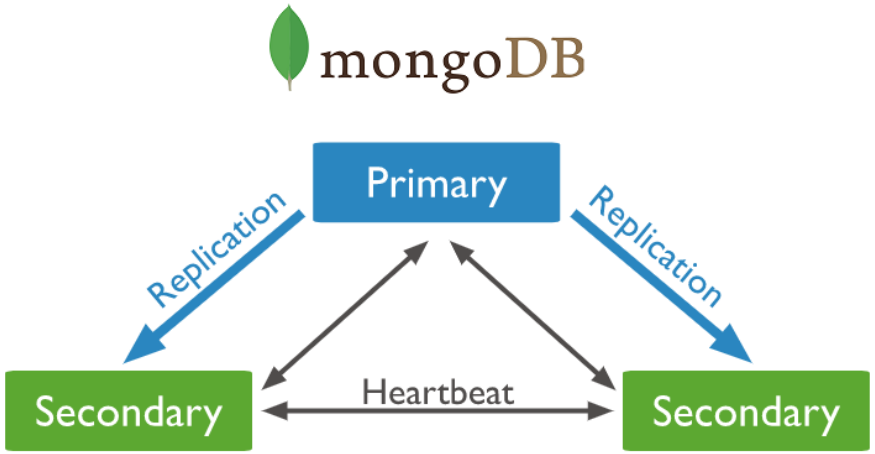
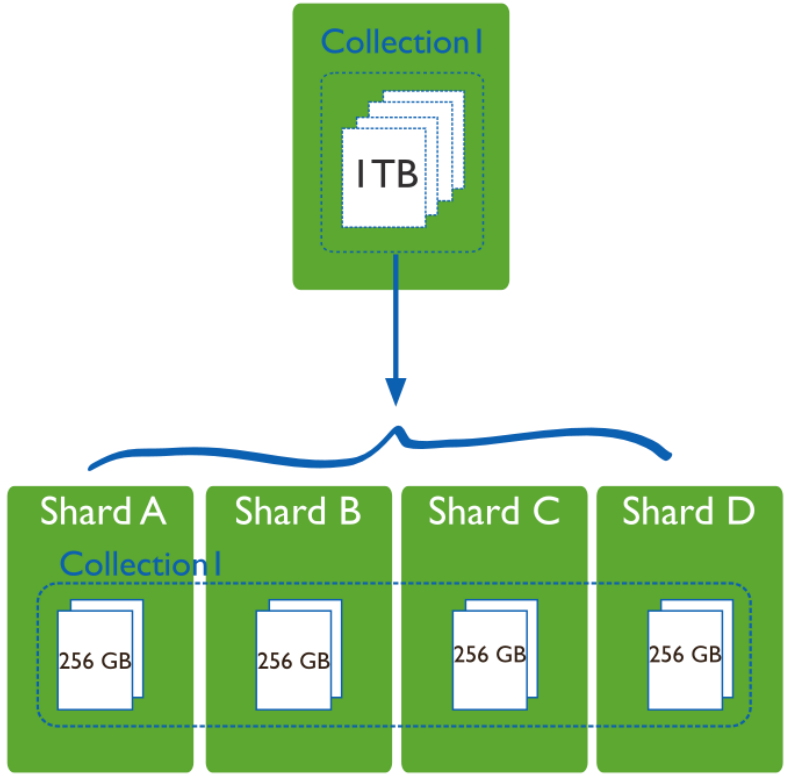
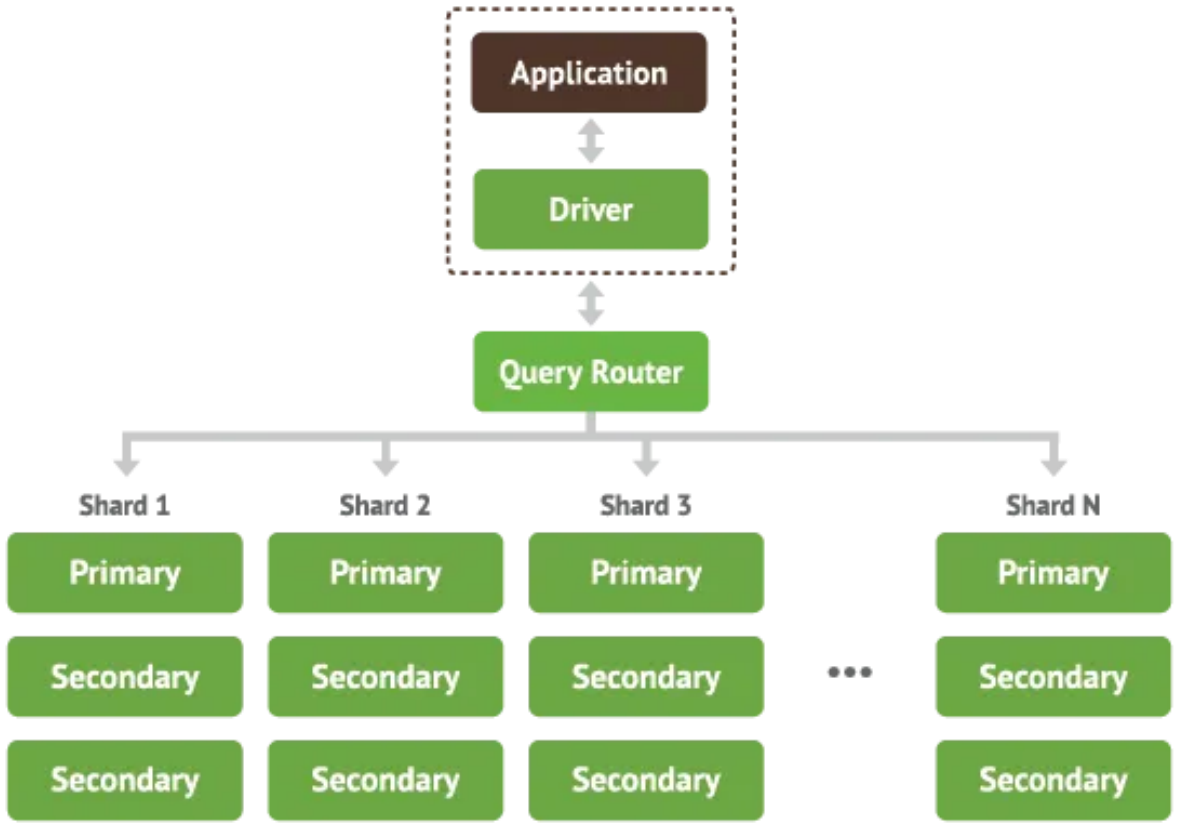
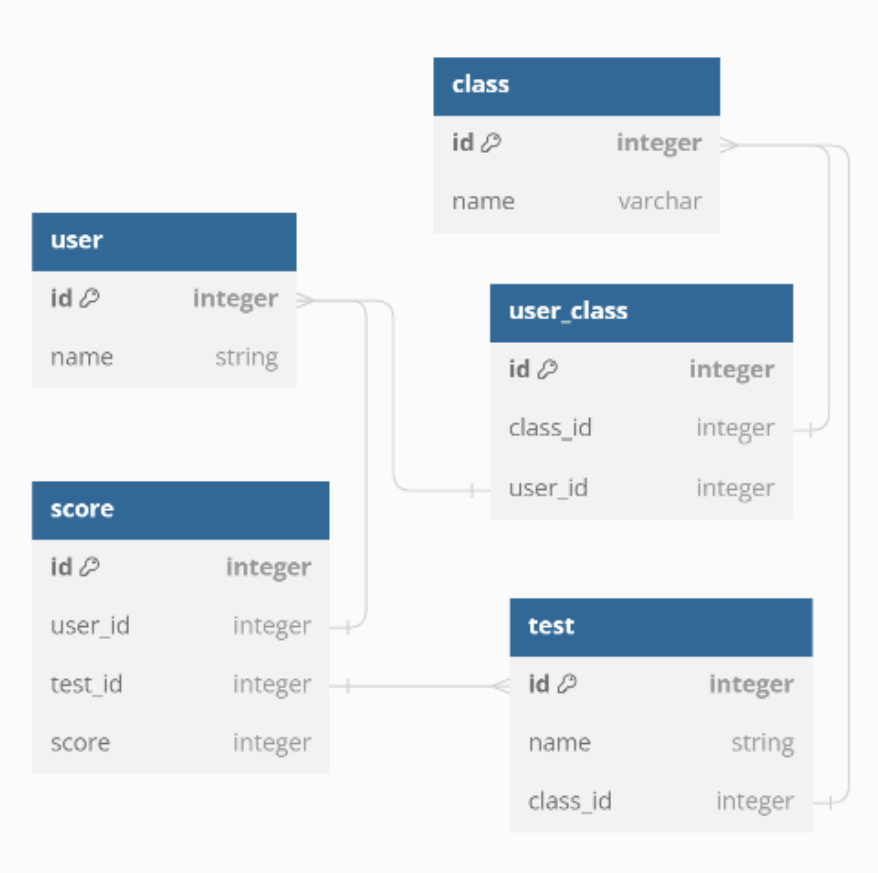
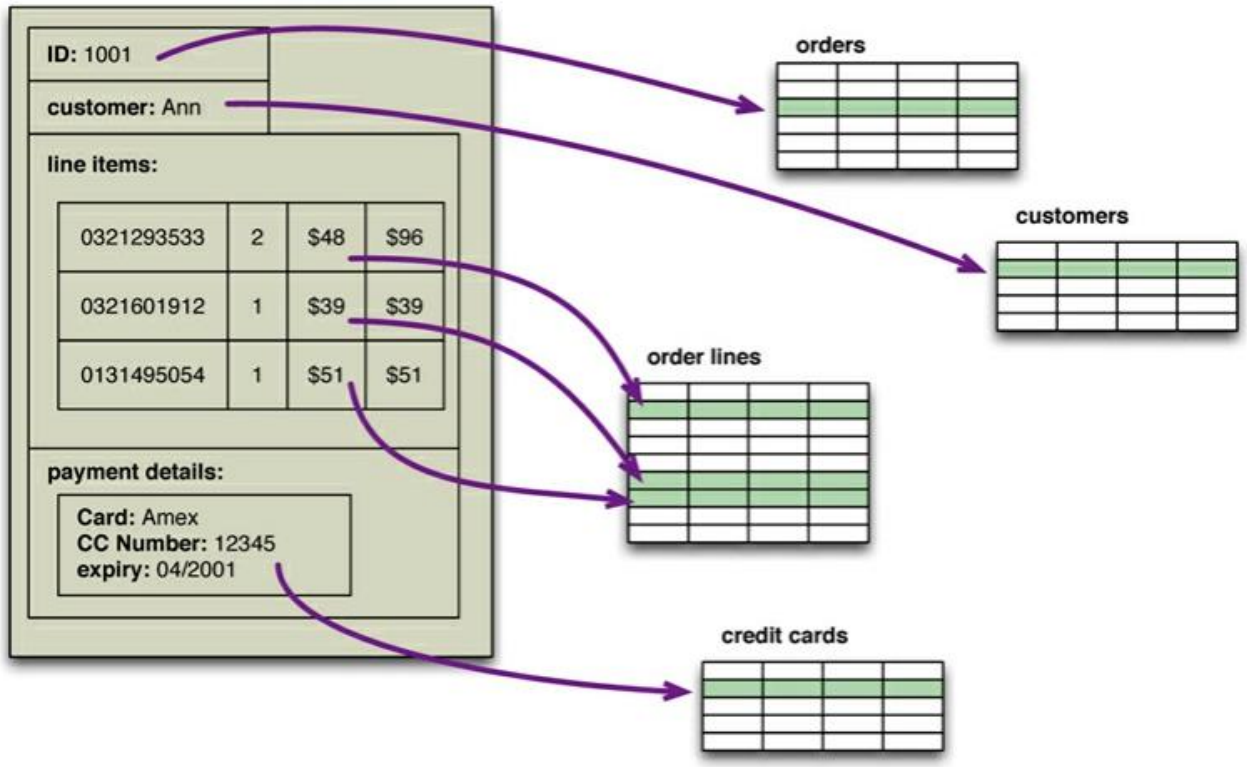
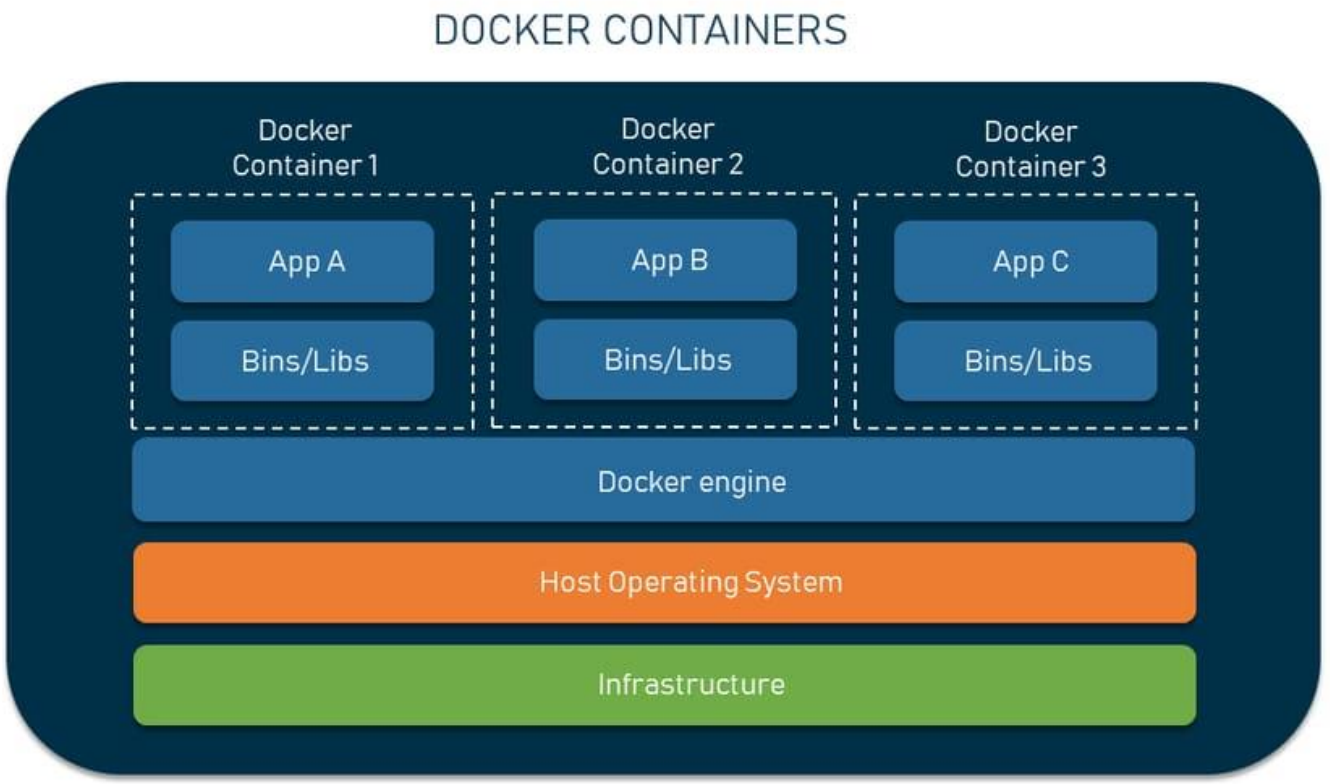
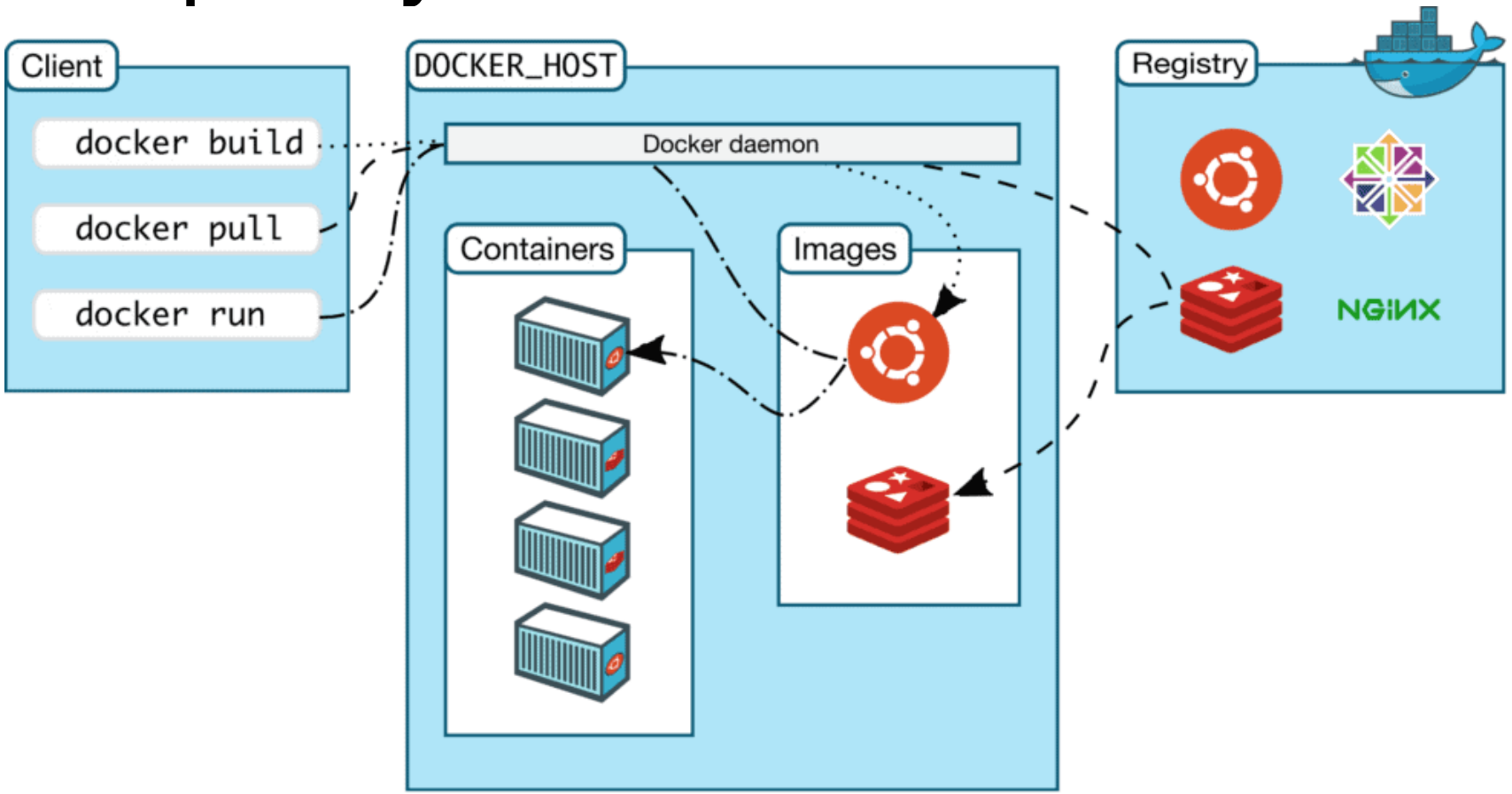
- Transform the following tree into a valid JSON file (by hand)



JSON - Exercise 2

- Transform the following text into a JSON file (by hand)
- Ingredients
  - Beef, chopped: 450g
  - Onions, minced: 350g
  - Carrot, chopped: 100g
  - Potato, chopped: 150g
  - Water: 500ml
  - Golden Curry Sauce Mix: 92g
- Directions
  - Stir-fry meat and vegetables with oil in a large skillet on medium heat for approx. 5 min.
  - Add water and bring to boil. Reduce heat, cover and simmer until ingredients are tender, approx. 15min.
  - Turn the heat off, break S&B Golden Curry Sauce Mix into pieces and add them to the skillet. Stir until sauce mixes are completely melted. Simmer approx. 5 min., stirring constantly.
  - Serve hot over rice or noodles.

JSON - Exercise 3

- Transform the following XML into a JSON
- ```
  <?xml version="1.0" encoding="UTF-8"?>
  <Race date="2020-12-15" name="Holiday Meet">
      <Course>
          <CourseName>The track</CourseName>
          <Address>Track road 123</Address>
      </Course>
      <Horses>
          <Horse name="Bonfire">
              <Value>5000</Value>
              <Birthdate>1998-05-01</Birthdate>
              <Gender>M</Gender>
          </Horse>
          <Horse name="Dobby">
              <Value>1000</Value>
              <Birthdate>2001-04-05</Birthdate>
              <Gender>F</Gender>
          </Horse>
      </Horses>
  </Race>
  ```
  ```

JSON: How to validate ?

- JSON Schema
  - Specification (2020): https://json-schema.org/specification

```
{
  "firstName": "John",
  "lastName": "Doe",
  "age": 21
}
```

JSON: How to validate ?

```
{
  "$id": "https://example.com/person.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "title": "Person",
  "type": "object",
  "properties": {
    "firstName": {
      "type": "string",
      "description": "The person's first name."
    },
    "lastName": {
      "type": "string",
      "description": "The person's last name."
    },
    "age": {
      "description": "Age in years which must be equal to or greater than zero.",
      "type": "integer",
      "minimum": 0
    }
  }
}
```

JSON: How to validate ?

```
{
  "fruits": [ "apple", "orange", "pear" ],
  "vegetables": [
    {
      "veggieName": "potato",
      "veggieLike": true
    },
    {
      "veggieName": "broccoli",
      "veggieLike": false
    }
  ]
}
```

JSON: How to validate ?

```
{
  "$id": "https://example.com/arrays.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "description": "Arrays of strings and objects",
  "title": "Arrays",
  "type": "object",
  "properties": {
    "fruits": {
      "type": "array",
      "items": {
        "type": "string"
      }
    },
    "vegetables": {
      "type": "array",
      "items": { "$ref": "#/$defs/veggie" }
    }
  }, […]
```

JSON: How to validate ?

```
[…]
"$defs": {
    "veggie": {
      "type": "object",
      "required": [ "veggieName", "veggieLike" ],
      "properties": {
        "veggieName": {
          "type": "string",
          "description": "The name of the vegetable."
        },
        "veggieLike": {
          "type": "boolean",
          "description": "Do I like this vegetable?"
        }
      }
    }
  }
}
```

JSON: How to validate ?

- Some other features
  - Regular expression
  - If-else
  - One of
  - All of
  - Any of
  - propertiesCount
  - Enumerations
  - …

JSON - Exercise 4

- Re-use the JSON created for *Exercise 3*, and create the associated JSON Schema to validate it
-

```
<?xml version="1.0" encoding="UTF-8"?>
<Race date="2020-12-15" name="Holiday Meet">
    <Course>
        <CourseName>The track</CourseName>
        <Address>Track road 123</Address>
    </Course>
    <Horses>
        <Horse name="Bonfire">
            <Value>5000</Value>
            <Birthdate>1998-05-01</Birthdate>
            <Gender>M</Gender>
        </Horse>
        <Horse name="Dobby">
            <Value>1000</Value>
            <Birthdate>2001-04-05</Birthdate>
            <Gender>F</Gender>
        </Horse>
    </Horses>
</Race>
```

(JSON) API

- Often used for RESTful API
- REST / Representational State Transfer
  - Set of architectural constraints, not a protocol nor a standard
  - Does not need to use JSON
- RESTful
  - API following the constraints of REST

RESTful API

- Constraints
  - Client-server architecture, with requests managed through HTTP
  - Stateless client-server communication
  - Cacheable data
  - Uniform interface
    - A resource must have only one logical URI
    - Self-descriptive messages with enough information for the client to process it
    - Hypertext/hypermedia to find all related information
    - Your endpoints should behave the same, allowing a developer to integrate it easily
  - Layered system
    - Client contacts API on server A, but if data or authorization is done on other servers it should be transparent

RESTful API


Endpoint
HTTP Method
HTTP Headers
Body
https://apiurl.com/review/new
POST
content-type: application/json
accept: application/json
authorization: Basic abase64string
{
"review" : {
"title" : "Great article!",
"description" : "So easy to follow.",
"rating" : 5
}
}
SitePoint

RESTful API


REVAALO
LABS
REST API Methods
GET
Receive information about an API resource
POST
Create an API resource
PUT
Update an API resource
DELETE
Delete an API resource

RESTful API

- `/users`
  - GET + url parameters
  - POST + body
- `/user/{id}`
  - GET
  - POST
  - PUT
  - PATCH
- `/orders`
- `/order/{id}`
- …

RESTful API - Why so common?



RESTful API - Why so common?

- Front-end for a desktop application
  - Then, for a mobile app
  - Then, custom front-end to deal with a legacy client
  - …
- You want your website to be more "dynamic" and smoother while loading new elements
  - Infinite scrolling on most websites
  - You want the client to automatically fetch the next data!
    - In HTML? XML? JSON? …?

RESTful API - Why so common?



Python web api

- demo

JSON - Exercise 5

- Use Python to create a RESTful API providing the various features
  - In memory "database"
  - (Shop) objects
    - Attributes: name, remaining quantity, price
    - List, create, update & delete
  - Orders
    - Attributes: object_id, quantity
    - List, create, update & delete
  - Apply some basic validations
    - quantity >= 0, name must be of length [4;100], birth date is a real date ...
- Provide a python script to test each endpoint

JSON - Credits & references

- Json
  - https://www.json.org/json-en.html
  - https://www.w3schools.com/js/js_json_intro.asp
- Json Schema
  - https://json-schema.org/specification
  - https://json-schema.org/learn/miscellaneous-examples

NoSQL Databases

Why?
How to use it ?
How to validate it ?

Why use a database?

- Efficient and persistent
- More flexible than using files
- Handle concurrent access
- Libraries to easily integrate with any programming language
- SQL / Relational databases share a lot of similarities
  - Many libraries handle PostgreSQL, MySQL, SQLite without any change
  - But each of them has custom features

Software and data

- Data usually lives longer than software
- Data is extremely valuable
  - Must be easy to interact with and stable
- Data should be at the center of the architecture

Relational databases

- Schema
- Tables
- Relationship between tables
- Easy querying using SQL
- Most common relational databases
  - MySQL / MariaDB
  - PostgreSQL
  - Microsoft SQL Server
  - SQLite (for local development)

Relational databases - Limitations

- Relationships
  - Indexes: RAM consumption, update overhead
  - High correlation between tables
- How to scale?
  - Vertically (Single server): Hardware limitations
  - Horizontally (multi-servers): How do deal with relationships efficiently?
  - Complex schema changes for large databases (1 TB+)
- We always manipulate (json) objects, so why use SQL at all?
  - NoSQL Databases

Scaling

- Vertical Scaling / Scale up
  - More powerful server
  - Architecture stays the same
- Horizontal Scaling / Scale out
  - Add more servers
  - Architecture needs to be designed for it
    - ! SQL Databases are still possible
    - All processes will not necessarily see the same state

Scaling - Database storage

- Database storage
  - Often one table (or database) = one file
  - Re-use deleted rows for new rows
  - Colocation of data is important
- Issues with "one file"
  - Backup
  - Schema changes
  - File system limitations
  - Handling of many deletions / updates: iops, lost disk space

Scaling - Database storage

- Issues with "many files"
  - Backup
  - Schema changes
  - File system limitations
  - Handling of many deletions / updates: iops
- Middle ground: partitions
  - User id [0…1000] → file 1
  - User id [1001…2000] → file 2

Scaling - Horizontal scaling

- Why do we want to scale?
  - Too much to write?
  - Too much to read?
  - Both?
- Lots of read operations
  - Read-replicas (1-3 are common)
  - Async replication with configurable delay
  - Software should be aware of it
  - Each server must still store everything



Scaling - Horizontal scaling

- Primary
  - Accepts Read & Write
- Secondary
  - Accepts Read
- Handling failure of the primary
  - Primary election
  - Software needs to know all nodes
  - Odd number of nodes is required



Scaling - Horizontal scaling

- Read replica
  - All the data still in each server
  - How to handle TBs?
  - Partitions at the cluster level: sharding



Scaling - Horizontal scaling

- Read replica + Sharding



Scaling - Horizontal scaling

- But JOINs ?
  - Highly normalized
- Example
  - Students registered to classes
  - Students have scores for tests
- How to scale?
  - Partition every table?
  - Latency ?
  - Bandwidth ?
  - RAM ?



Impedance mismatch

- User point of view
  - A single document
- Developer point of view
  - Multiple tables to manage
- Impedance mismatch
  - Difference between the relational model and the in-memory data structures



NoSQL Database - MongoDB

- Optimize storage for read
  - "Similar" to a Materialized View managed by yourself
- High freedom
  - Add/remove any field
  - Set any type
- Every user object is handled separately (no constraint)

```
{
  "id": 0,
  "name": "Alex",
  "classes": [
    {
      "id": 42,
      "name": "Programming"
    }
  ],
  "scores": [
    {
      "test": {
        "id": 25,
        "name": "1st test"
      },
      "score": 20
    }
  ]
}
```

Relational Databases - Issues

- Conversion of data between end-user and data storage
- Reconstruction of data from tables
- Fixed data model
- Relational databases forces columns of a specific type (generally)
- Scaling issues
- Complicated searching in relational database
- But
  - SQL
  - Many features to do "anything" (streaming, ...)
  - Schema enforcement

Non-Relational Databases - Issues

- Almost no data type enforcement
- Data Model is extremely free (few available constraints)
- Limited set of features
- Specific languages
- Transactions are generally not supported natively
- But
  - Designed for scalability
  - Data type freedom (media, text, json, …)
  - Do a few things efficiently
  - Schema enforcement

Which one to choose?

- RDBMS are powerful and stable
- NoSQL DBs are specialized and easily scalable
- Many architectures use both

Which one to choose?

- MongoDB
- Cassandra
- Couchdb
- Hbase
- Redis
- Neo4j
- Amazon AWS
  - RDS
  - DynamoDB
  - …
- …

How to quickly test a database?

- Manual installation of redis
  - https://redis.io/docs/latest/operate/oss_and_stack/install/install-redis/
  - Centos
    - `sudo yum install redis && sudo system start redis`
  - How to remove all files?
    - `sudo yum uninstall redis`
    - But… some files will remain
    - How to quickly test various versions and make sure all dependencies are properly removed / installed?
  - How to make sure all the applications are installed together for your software?
    - Bash script but how to handle updates? Deletion? …

How to quickly test a database?



altexsoft

How to quickly test a database?



How to create a docker image?

- Dockerfile
  - ```
    FROM python:3.8-slim-buster
    WORKDIR /python-docker
    COPY requirements.txt requirements.txt
    RUN pip3 install -r requirements.txt
    COPY . .
    CMD [ "python3", "-m" , "flask", "run", "--host=0.0.0.0"]
    ```
  - `docker build -t flask:0.1.0 .`
- Redis
  - https://github.com/redis/docker-library-redis/blob/master/Dockerfile.template
- MongoDB
  - https://github.com/docker-library/mongo/blob/master/Dockerfile-windows.template
    ```

How to quickly test a database?

- Install docker
  - https://docs.docker.com/engine/install/
  - `docker run --name some-redis -d redis`
- Install docker-compose
  - https://docs.docker.com/compose/install/
  - Create a `docker-compose.yml`

```yaml
version: '3'

services:
  redis:
    # https://hub.docker.com/_/redis
    image: redis:7.4.0
    container_name: redis
    restart: unless-stopped
```

How to quickly test a database?

- Interact with your processes
  - `docker compose up redis`
  - `docker compose up redis -d`
  - `docker compose down`
  - `docker logs -f redis`
- How to enter a docker?
  - `docker exec redis -it /bin/bash`
- How to keep your data?
  - Volumes
- How to provide some configuration files?
  - Volumes
- How to expose a port?
  - Ports

NoSQL - Exercise 1

- Create a docker-compose which contains the following services (single node)
  - Postgres
  - MongoDB
  - Cassandra
  - Couchdb
  - Redis

Data models

- Data model: model through which we see, perceive and manipulate the data
  - How to use the data
  - Different from the storage model: how the database stores and manipulates the data internally
- Storage model must be kept in mind for performance optimization
- In the Relational data model, data consists of table and rows
  - Columns with values that can reference other rows
- In the non-relational world, it depends

NoSQL Databases - Types

- There are 4 main categories of NoSQL DBs
  - Key-value
  - Column oriented
  - Document oriented
  - Graph oriented
- Key-value, column oriented, and document oriented share a common characteristic of their data models
  - aggregate orientation
- Some databases can handle multiple of those models

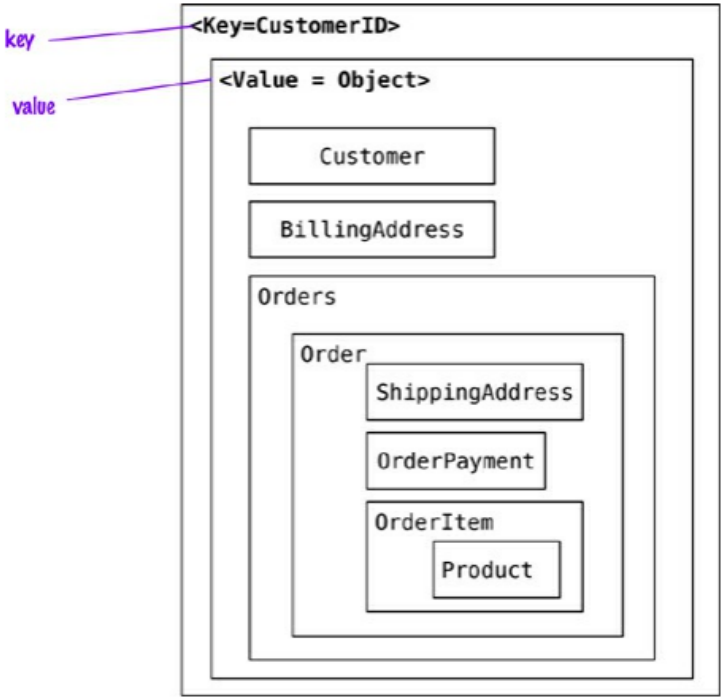
Aggregates

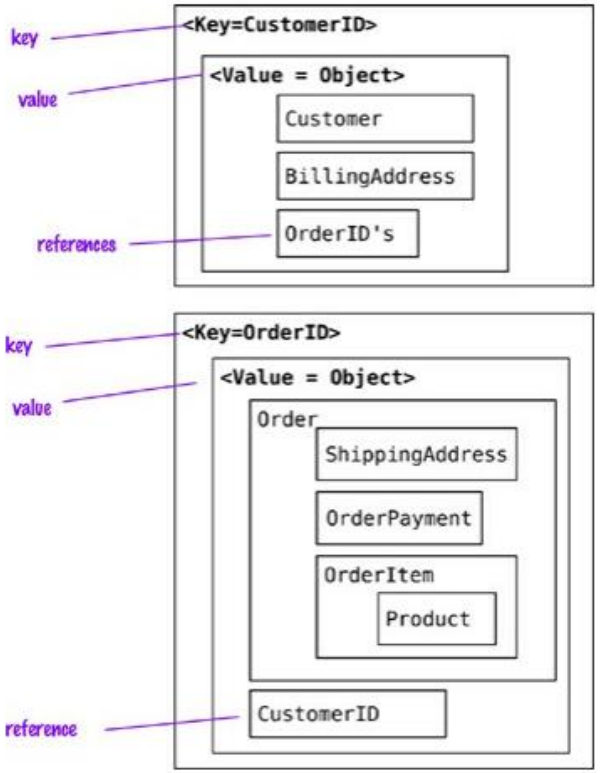
- A collection of related objects that we wish to treat as a unit
- A unit for data manipulation and management of consistency
- A natural unit for cluster data management

```
{
  "id": 0,
  "name": "Alex",
  "classes": [
    {
      "id": 42,
      "name": "Programming"
    }
  ],
  "scores": [
    {
      "test": {
        "id": 25,
        "name": "1st test"
      },
      "score": 20
    }
  ]
}
```

Modeling for data access

- When modeling data aggregates we need to consider how the data is going to be accessed





Modeling for data access

- Column and graph oriented

Row-oriented

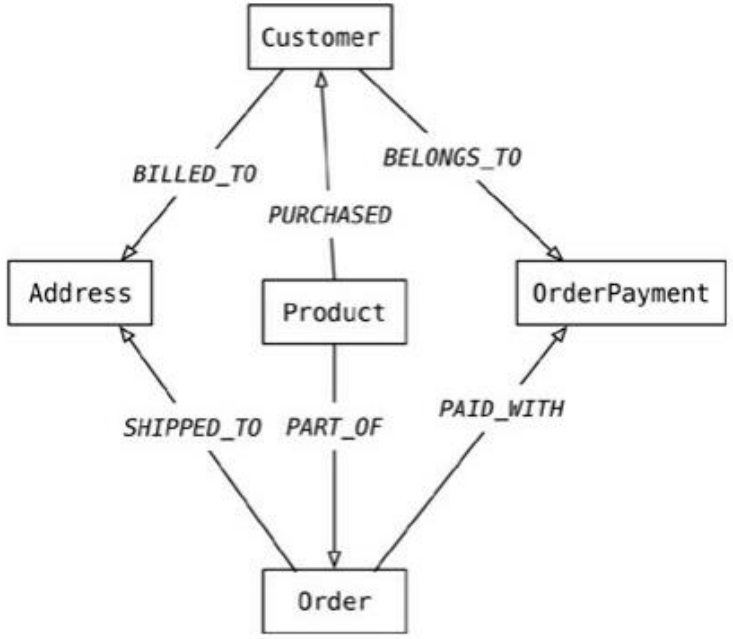
ID	Name	Grade	GPA
001	John	Senior	4.00
002	Karen	Freshman	3.67
003	Bill	Junior	3.33

Column-oriented

Name	ID
John	001
Karen	002
Bill	003

Grade	ID
Senior	001
Freshman	002
Junior	003

GPA	ID
4.00	001
3.67	002
3.33	003



Schemaless databases

- While relational databases must have a predefined schema, NoSQL databases are often - but not always - schemaless
  - Freedom and flexibility
  - Easy non uniform data handling
    - RDB would need a lot of NULL columns
- Schemaless on paper, but you should ***really*** define a model
  - `qty, quantity, Quantity`
  - `time, time_ms, time_s`
  - You need an implicit schema, at the application level
  - Do not forget about data migrations!
    - Do you really want to maintain the data model of 5 years ago & 10 iterations ago?

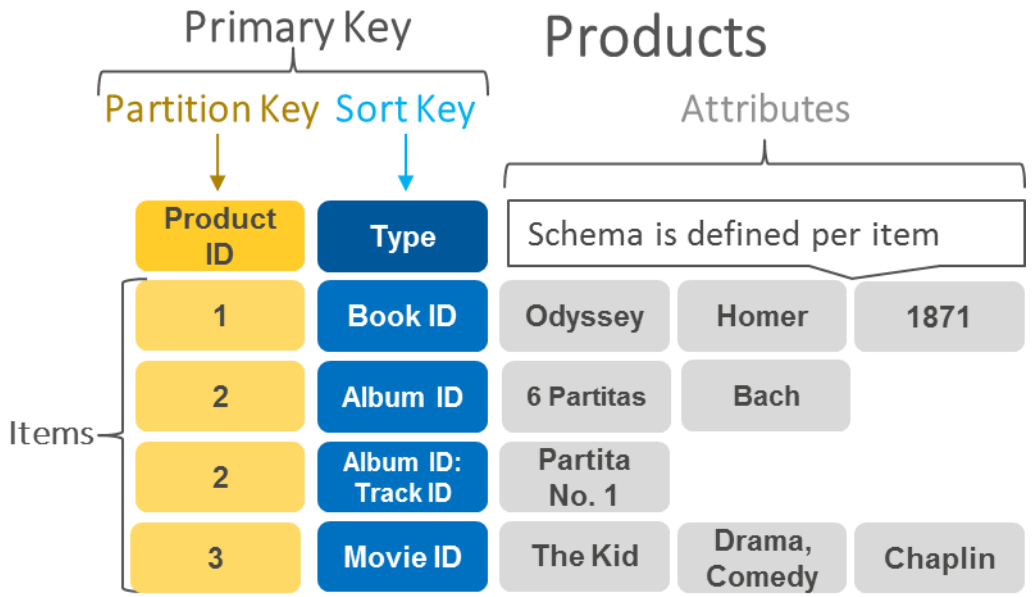
Key-Value Databases

- O(1) in time execution to read/write a record
  - Direct mapping from key to storage / RAM
- Persistence is optional
  - Often used as non-persistent cache
- Extremely inefficient to filter on keys & values
  - You should know the key or at least a key prefix
- Examples
  - Redis
  - Memcached
  - AWS DynamoDB
  - HBase

Key	Value
K1	AAA,BBB,CCC
K2	AAA,BBB
K3	AAA,DDD
K4	AAA,2,01/01/2015
K5	3,ZZZ,5623

Key-Value Databases - AWS DynamoDB

- Software-as-a-service
- Often used as persistent storage
- Infinitely scalable (PB+)
- Secondary indexes
- Filter is possible but you should use indexes
- Some constraints
  - Object size < 400KB
  - Query/s is dependent on the partitions



Key-Value Databases - AWS DynamoDB

- NoSQL databases
  - Servers & Clusters
- DynamoDB
  - Partitions
  - Basic premise: There is a way to shard data that's horizontally scalable
  - Resources are allocated dynamically
- Partition key
  - Distribute traffic
  - Need high cardinality
  - Evenly spread

Key-Value Databases - AWS DynamoDB

- Store data
  - ```
    import boto3

    dynamodb = boto3.resource('dynamodb')
    table = dynamodb.Table('YourTableName')

    table.put_item(
        Item={
            'pk': 'id#1',
            'sk': 'cart#123',
            'name': 'SomeName',
            'inventory': 500,
        }
    )
    ```
    ```

Key-Value Databases - AWS DynamoDB

- Query data
  - ```
    import boto3
    from boto3.dynamodb.conditions import Key, Attr

    dynamodb = boto3.resource('dynamodb')
    table = dynamodb.Table('YourTableName')

    response = table.query(
        KeyConditionExpression=Key('pk').eq('id#1') &
    Key('sk').begins_with('cart#'),
        FilterExpression=Attr('name').eq('SomeName')
    )
    ```
    ```

Key-Value Databases - AWS DynamoDB

- CLI
  - `aws dynamodb create-table --table-name Music --attribute-definitions AttributeName=Artist,AttributeType=S AttributeName=SongTitle,AttributeType=S --key-schema AttributeName=Artist,KeyType=HASH AttributeName=SongTitle,KeyType=RANGE --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5`
  - `aws dynamodb put-item --table-name Music --item file://item.json`
  - `aws dynamodb update-item --table-name Music --key '{ "Artist": {"S": "Acme Band"}, "SongTitle": {"S": "Happy Day"}}' --update-expression "SET AlbumTitle = :newval" --expression-attribute-values '{":newval":{"S":"Updated Album Title"}}' --return-values ALL_NEW`
  - `aws dynamodb get-item --table-name Music --item file://item.json`
  - `aws dynamodb delete-item --table-name Music --key file://key.json`
  - `aws dynamodb query --table-name Music --key-condition-expression "ArtistName=:Artist and SongName=:Songtitle"`

Key-Value Databases - AWS S3

- Software-as-a-service
- Persistent storage
  - S3 Standard
  - S3 Intelligent-Tiering
  - S3 One Zone-IA
  - S3 Glacier
  - S3 Outposts
  - …
- Durability: 99.999999999 %
- Infinitely scalable (PB+)
- Object size < 5TB
  - Cannot append / update part of the file
- Some constraints
  - Object size < 5 TB
  - Query/s is dependent on the prefix

```
s3://ecam-2024/prefix1/prefix2/picture.jpg
s3://ecam-backup/2024/huge-backup.zip
```

Key-Value Databases - AWS S3

- Extremely integrated with all of AWS services
- Documentation > 3000 pages
  - https://docs.aws.amazon.com/pdfs/AmazonS3/latest/userguide/s3-userguide.pdf
- Some features
  - Full websites (front-end)
  - Backup
  - Logs storage
  - File sharing
  - Versioning
  - Time To Live
  - Encryption
  - Security compliance: PCI DSS, HIPAA, ...

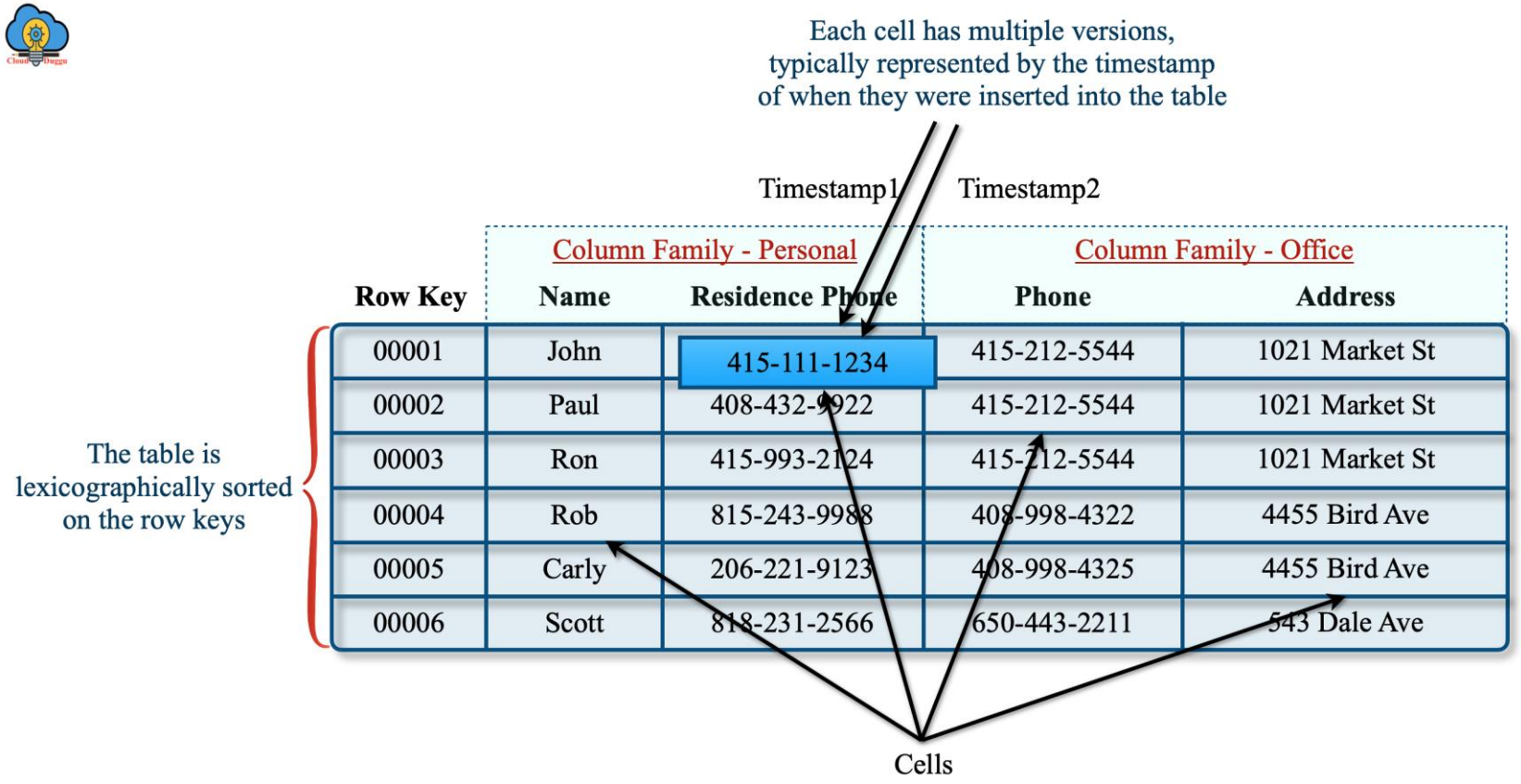
Key-Value Databases - AWS S3

- Object is constituted of
  - Key
  - Version ID
  - Value (up to 5TB)
  - Metadata
  - Subresources
  - Access Control Information
  - Tags

Key-Value Databases - AWS S3

- CLI
  - `aws s3api get-object --bucket text-content --key dir/my_images.tar.bz2 my_images.tar.bz2`
  - `aws s3api get-object --bucket text-content --key dir/my_data --range bytes=8888-9999 my_data_range`
  - `aws s3api put-object --bucket text-content --key dir-1/my_images.tar.bz2 --body my_images.tar.bz2`
  - `aws s3api list-objects-v2 --bucket my-bucket`

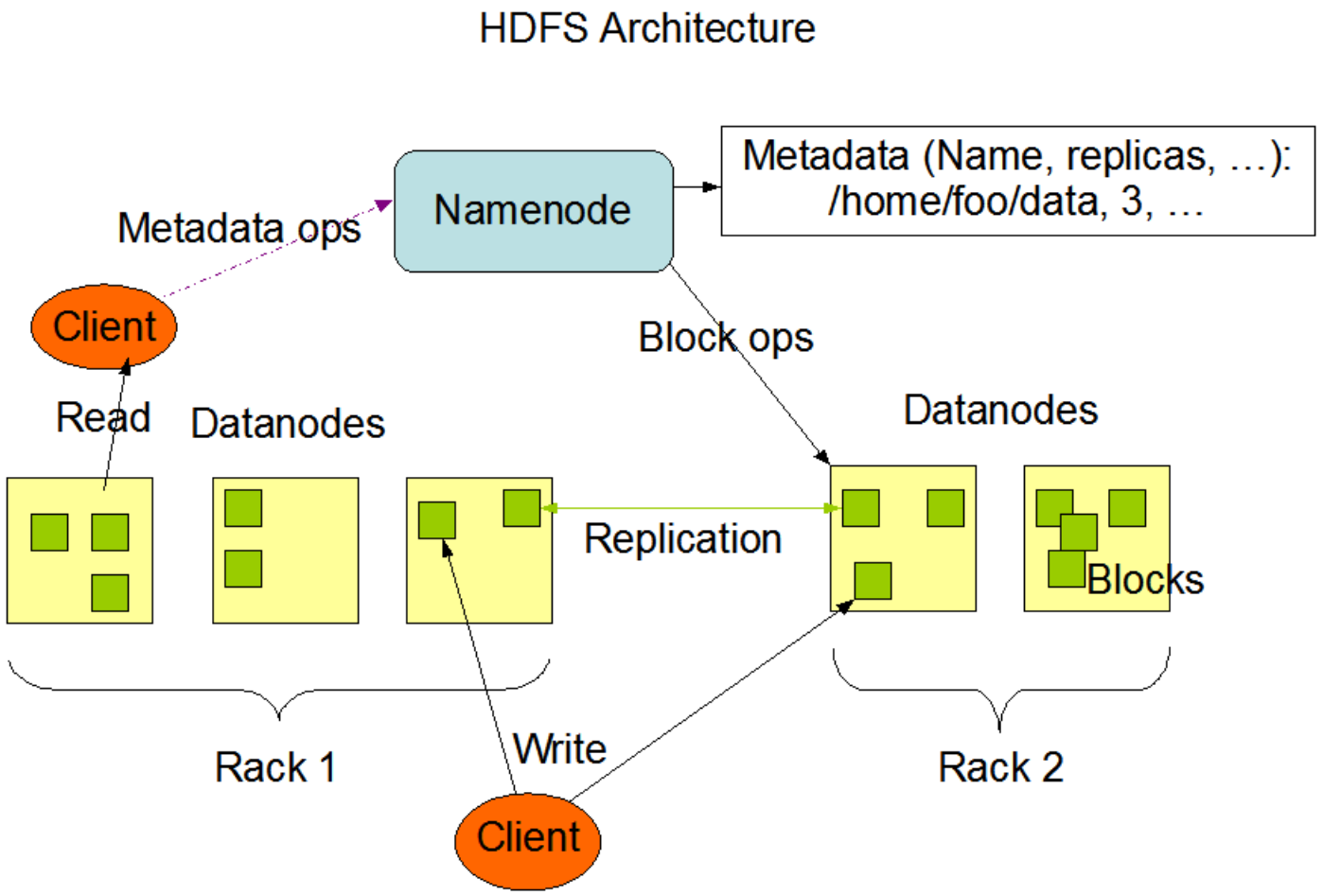
Key-Value Databases - HBase



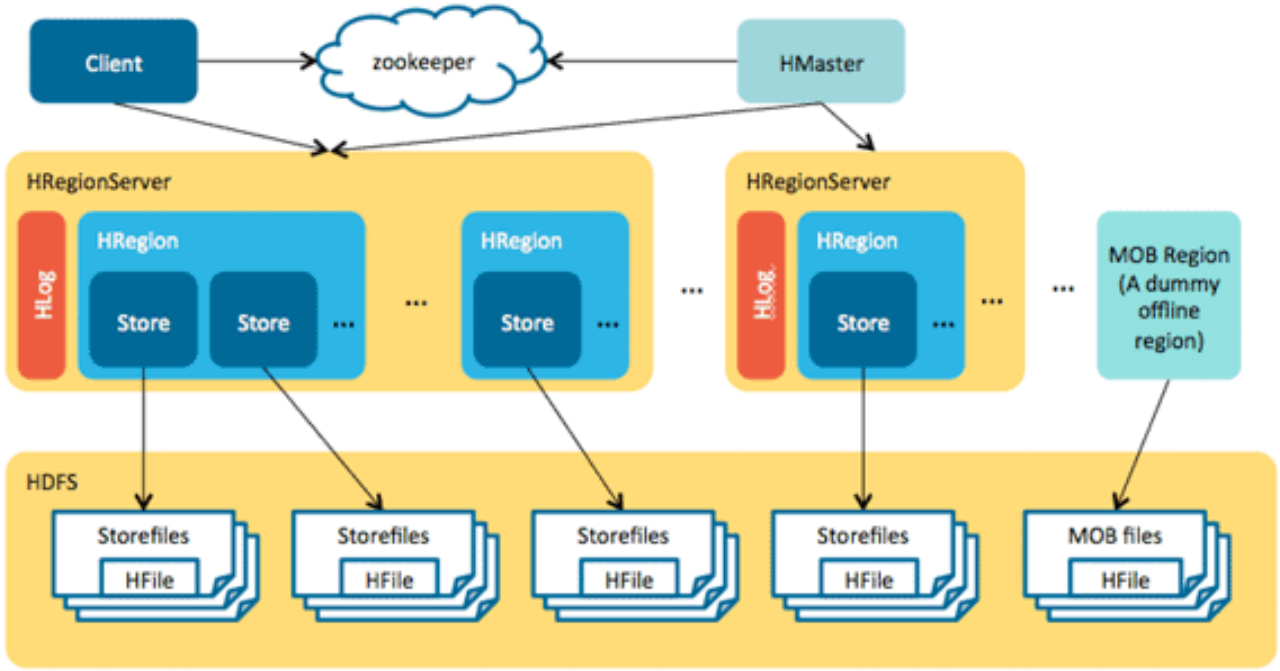
Key-Value Databases - HBase

- Most of the time within the Hadoop ecosystem
  - Highly distributed environment with Map-Reduce at its core
    - Massively parallel computing framework (10 000+ nodes)
- Family Columns: Column oriented
- HDFS
  - Files cannot be updated
  - Files should be minimum 64/128 MB in size
- Extremely efficient RAM wise
  - All keys do not need to be in RAM
- Filtering is possible, but inefficient

Key-Value Databases - HBase



Key-Value Databases - HBase

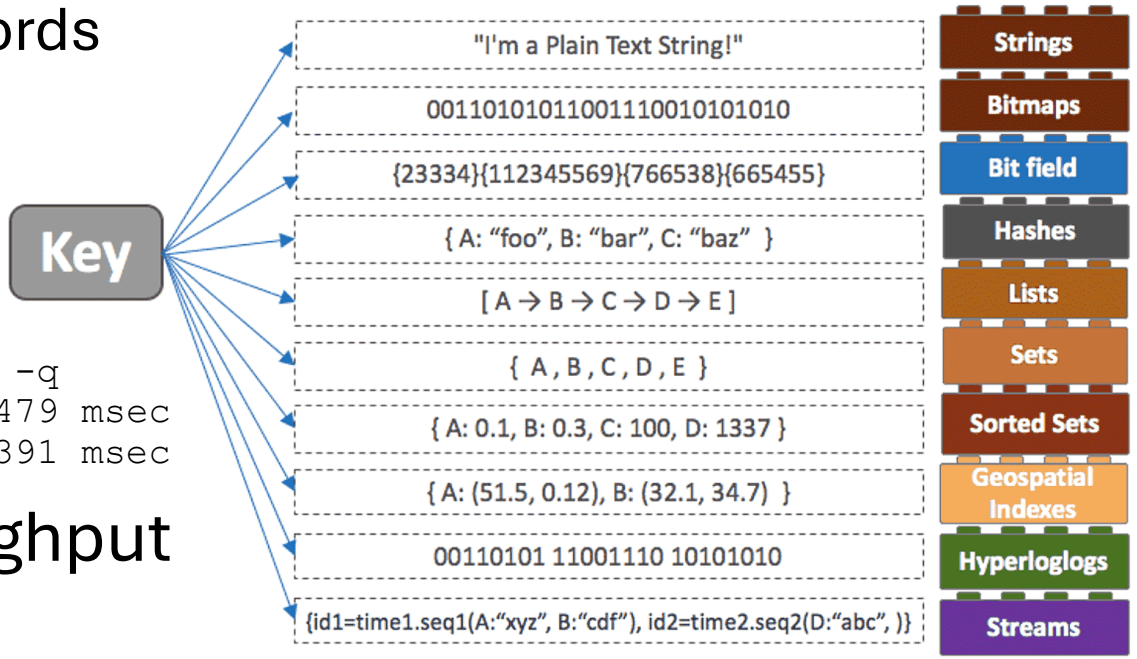
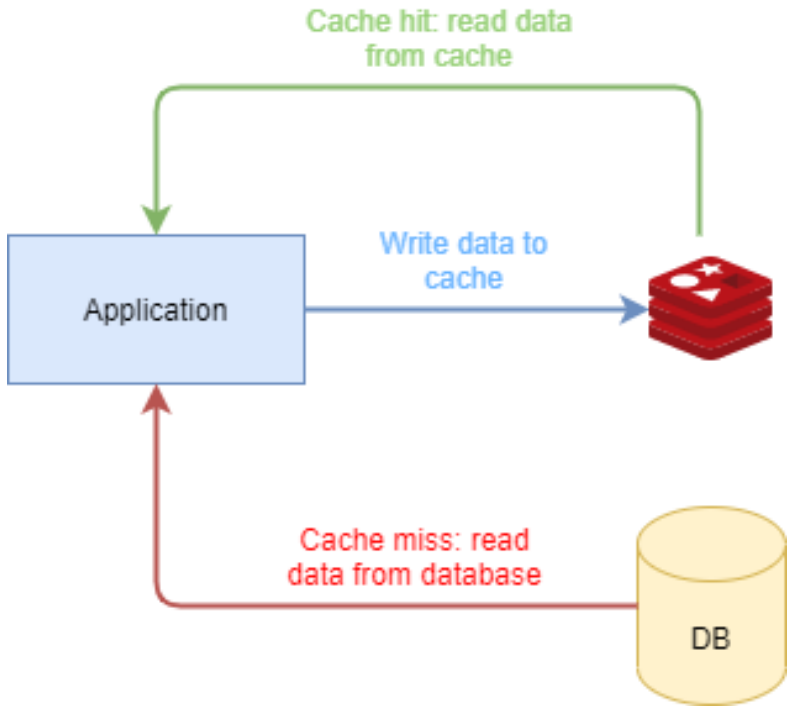
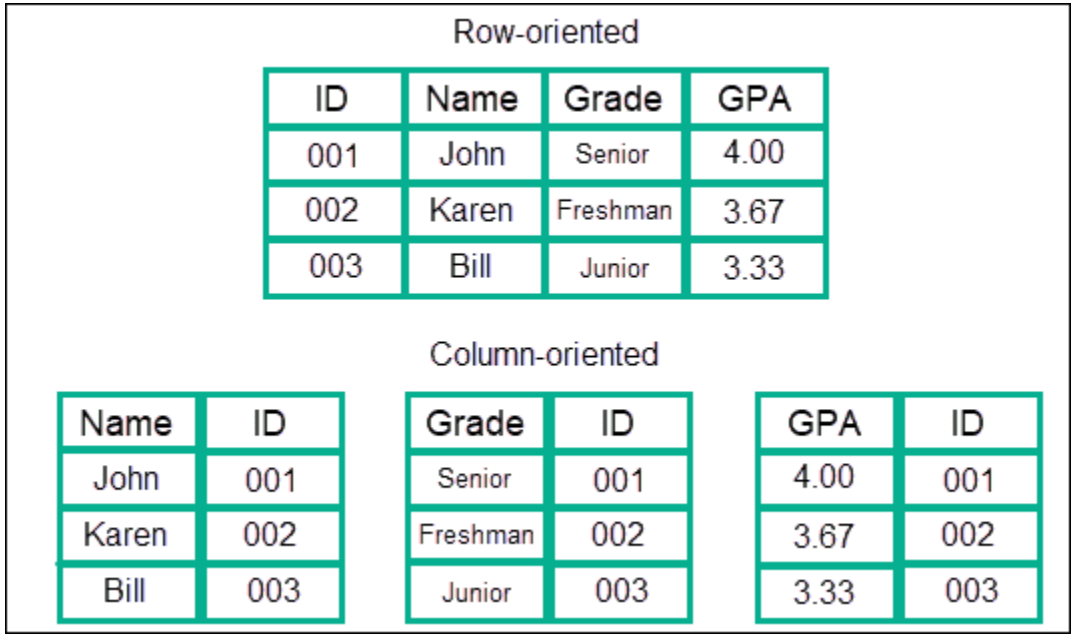
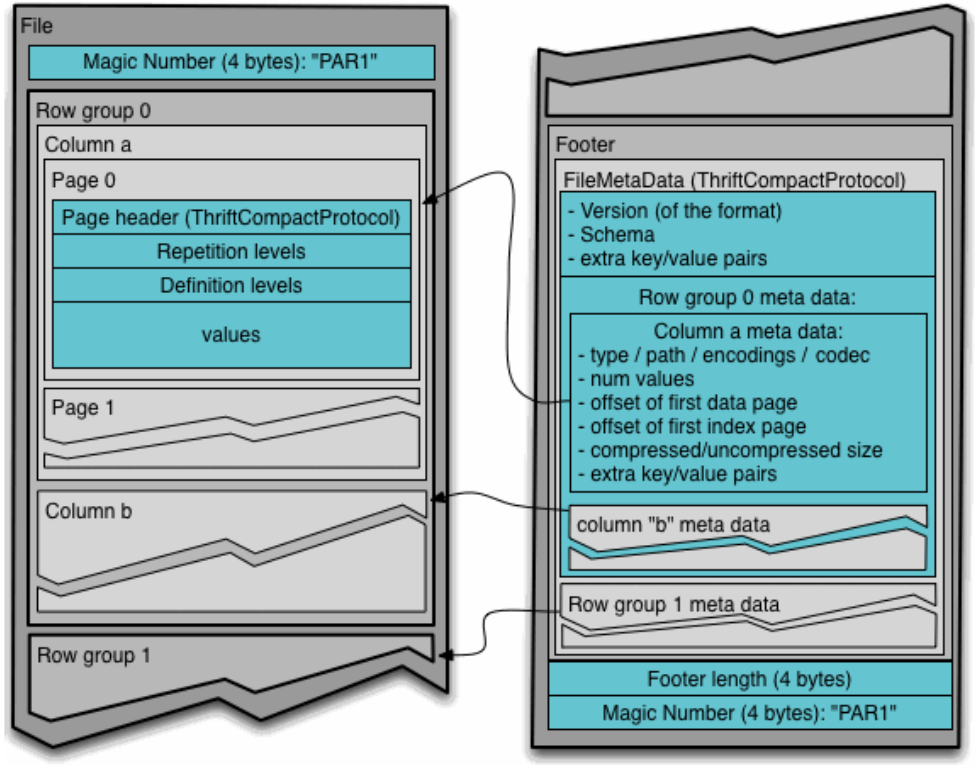


Key-Value Databases - HBase

- CLI
  - `create 'emp', 'personal data', 'professional data'`
  - `put '<table name>', 'row1', '<colfamily:colname>','<value>'`
  - `put 'table name', 'row','Column family:column name','new value'`
  - `get '<table name>', 'row1'`
  - `delete '<table name>', '<row>', '<column name >', '<timestamp>'`
  - `scan '<table name>'`

Key-Value Databases - Redis

- Frequently used as cache
  - Time-to-live feature to drop old records
  - Can be persisted to disk
  - Data must fit in RAM
- Often deployed on a single node
  - ```
    redis-benchmark -n 1000000 -t set,get -P 16 -q
    SET: 1536098.25 requests per second, p50=0.479 msec
    GET: 1811594.25 requests per second, p50=0.391 msec
    ```
- Extremely low latency, high throughput

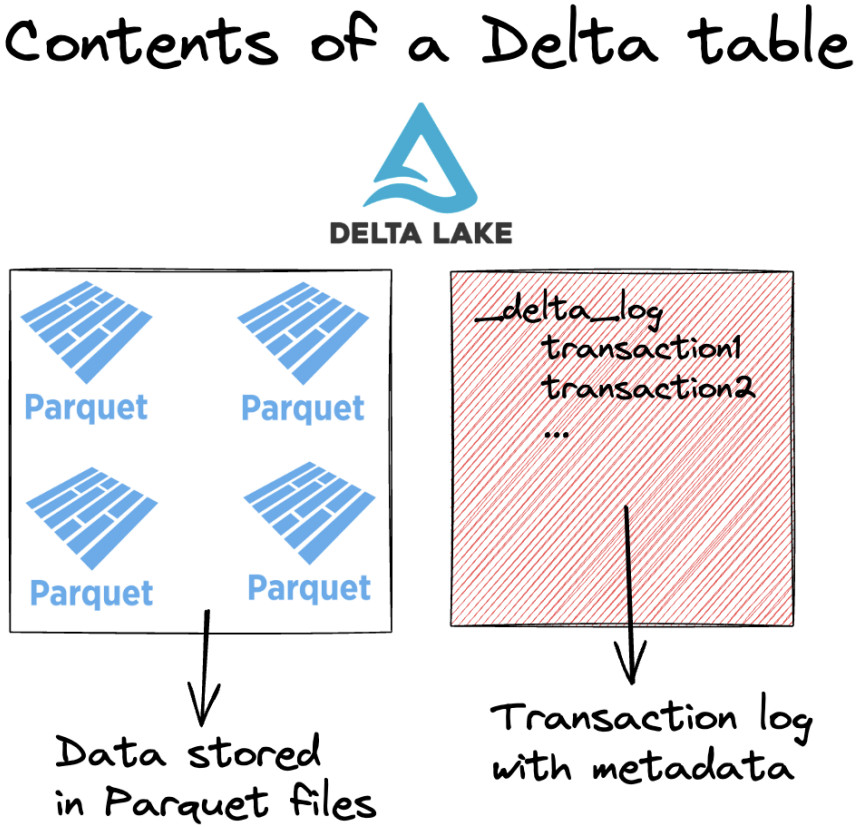
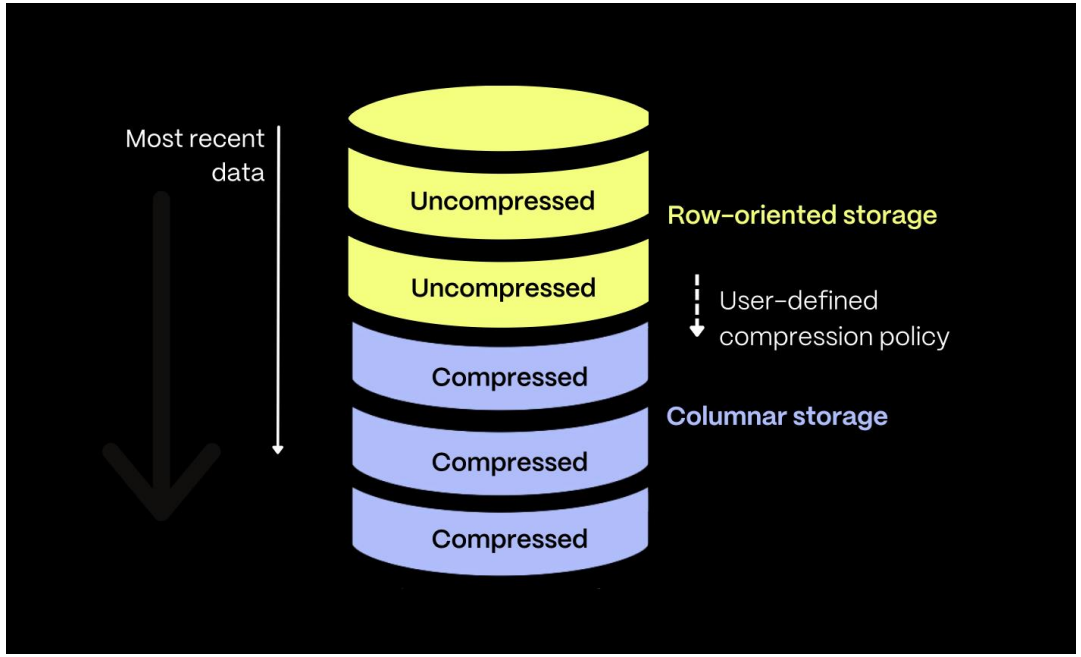
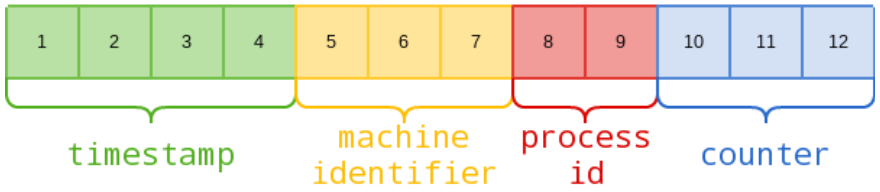
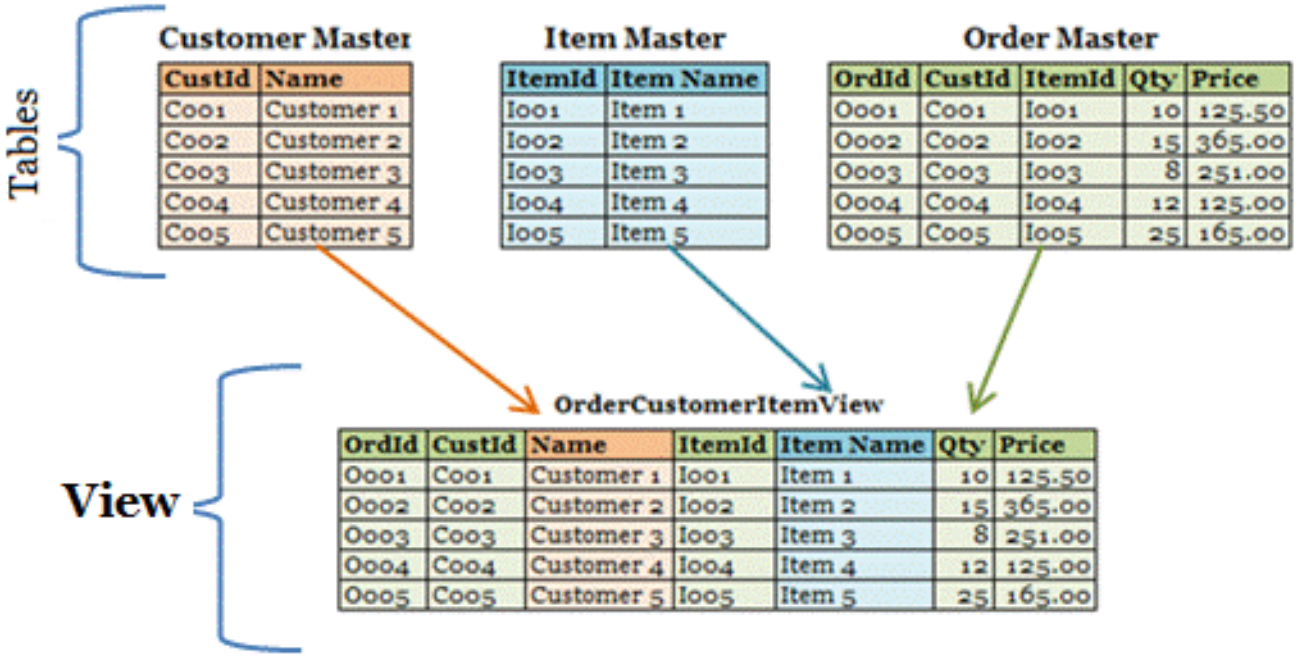
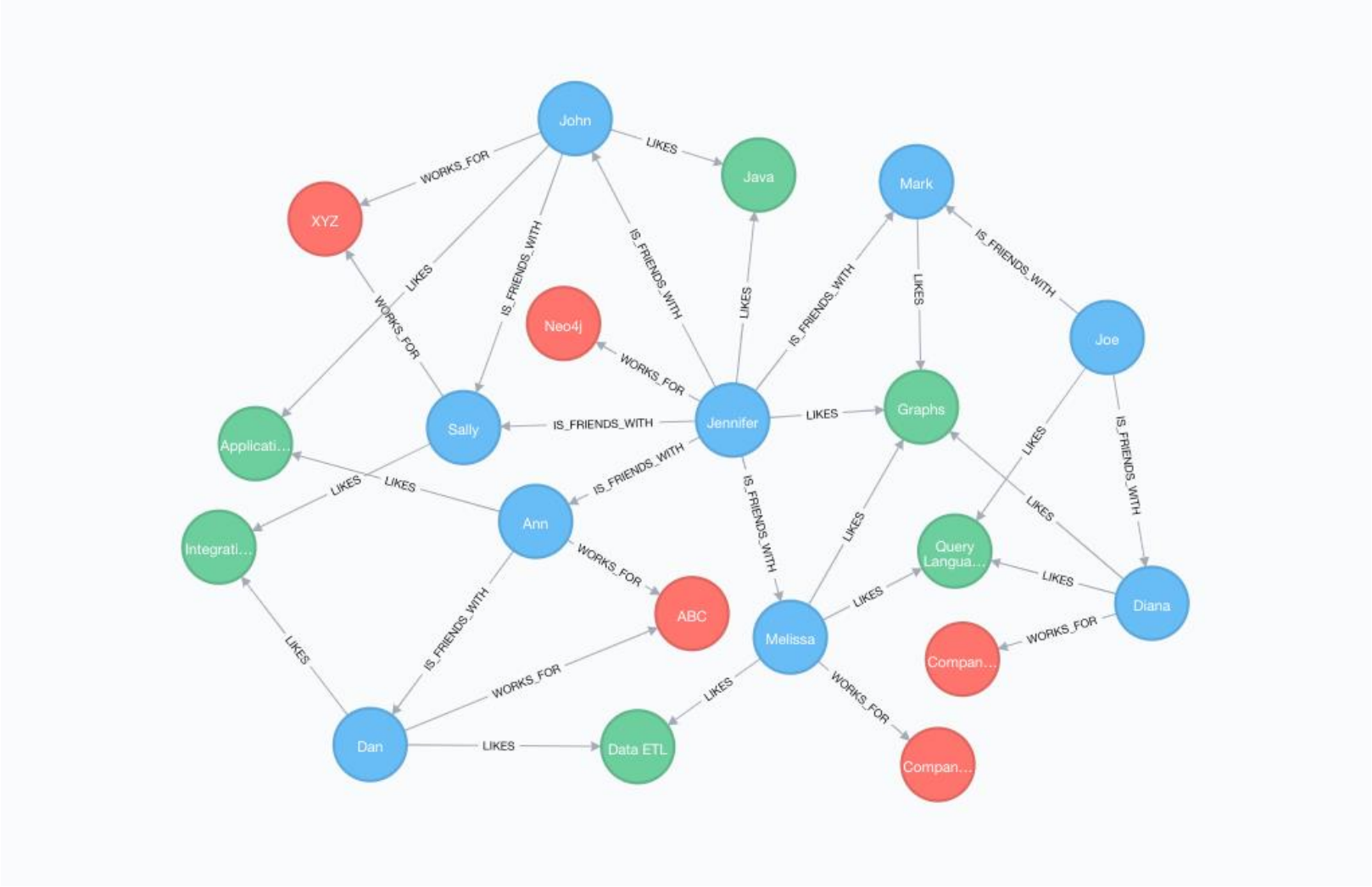
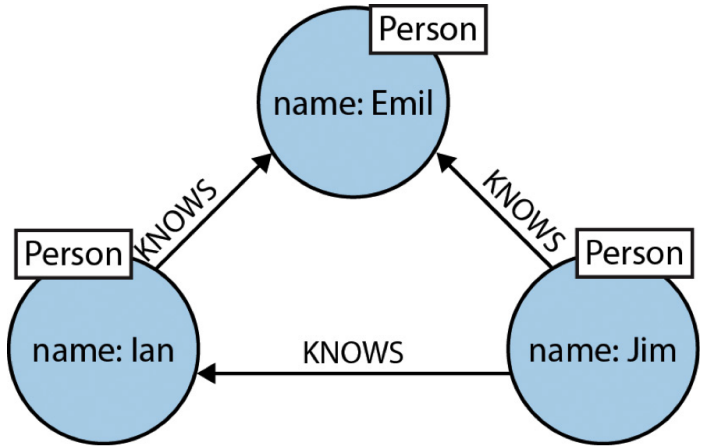

    ```

Key-Value Databases - Redis



Key-Value Databases - Redis

- Read-through caching
  - Ask the cache the data, if not available the cache loads it itself from the database
- Write-through caching
  - Always update the data in the underlying database, then in the cache
- Write-behind caching
  - Persistence to data source is delayed
- Refresh-ahead caching
  - Refresh old data to avoid a cache miss

Key-Value Databases - Redis

- Pub/sub system

- CLI
  - `SET anotherkey "will expire in a minute" EX 60`
  - `GET key`
  - `DEL key1 key2 key3`
  - `SCAN`

Key-Value Databases - Storage

- Access to a single key
- Access to multiple keys
  - Colocated ?
  - In RAM?
- Are HDD good enough?
  - 200-300 MB/s
  - 200-300 iops
  - How to achieve > 200 queries/s ?

vs

Key-Value Databases - Storage

- Are SSD the solution?
  - SSD SATA
  - SSD SATA PCIe
  - SSD NVMe PCIe 5.0 x4

- Crucial T705 2TB
  - Read: 10 GB/s, Write: 11 GB/s
  - 1 300 000 IOPS
  - Read 4k: 80 MB/s, Write 4k: 270MB/s
- Are SSD good enough?

Key-Value Databases - Storage

- Are SSD good enough?
- RAM: 120ns
  - L3 Cache: 12.9ns
  - L2 Cache: 2.8ns
  - L1 Cache: 0.9ns
  - Register: 0.3ns
- Colocation of data is not only for the persistent storage

Exercise 1 - Application

- Develop a python HTTP API for a shopping cart
  - ! It is going to be the base of all subsequent exercises
- Features
  1. Create the necessary tables for the application (at startup)
  2. Create x users
     1. Example: `POST /users` with `{"quantity": 10000}`
  3. Create x objects having between (y; z) stock availability and a price between (p1; p2)
     1. Example: `POST /objects` with `{"quantity_min": 100, "quantity_max": 500, "price_min": 20.0, "price_max": 30.0}`
  4. Create x orders spread across y users and z objects
- Use sqlite
  - No need to use SQLAlchemy or other ORM (unless you want to), you can use raw sql

Exercise 2 - Redis

- Connect a Redis instance to your application
- New features
  1. For a given user id x, display the number of distinct users having bought any identical object with you
  2. If the previous command is executed twice for the same user, make sure the second execution gives back the result in < 10ms
     - This timing should be respected even if your database contains millions of users & objects.
  3. Make sure the first execution for any user gives back the result in < 10ms as well
     - This timing should be respected even if your database contains millions of users & objects.
     - Hint: You might want to pre-compute the values

Deliverables

- For this course, you can submit your work on Friday 11/10 at 23:59 the latest
  - Forgot attached file: 0/20
  - Wrong file sent: 0/20
  - Late submission: 0/20
  - "Copy-paste" from another student: 0/20
    - Attempt to hide: 0/20 + special attention to every submission (including project)
  - If you already sent the "Parquet" exercise previously, send it again with this course zip, otherwise it will not be graded

Column oriented databases

- Data storage model of relational database
  - Row ~= file line : every column value
  - Alter a table?
    - Updates all rows
  - Need to access a single column?
    - Must read all columns of the target row
    - Or at least, data is not contiguous so iops to consider
- What about timeseries data ?
  - `series_id | application_date | last_update | value | insertion_date | creator | confidence`
  - * billions data records
  - 200 bytes per record, but only interested in 3 columns (30 bytes)? Too bad

Column oriented Databases

- Maybe we could store data per column



- Only useful if you avoid doing the equivalent of a SELECT *

Column oriented Databases

- OLTP
  - Online Transaction Processing
  - Commonly row-oriented
    - CSV
    - PostgreSQL
- OLAP
  - Online Analytical Processing
  - Commonly column-oriented
    - Apache Parquet
    - Apache Kudu
    - Google Cloud BigTable
    - Amazon Redshift

Column oriented Databases - Parquet

- Not really a database on its own
- Free and open source data storage
  - Similar to ORC and RCFile
- Efficient data compression
  - Values often share the same pattern
  - IDs, if sorted, have an identical prefix
  - Specialized encoding & compression per column
- Widely supported, and often used by other databases
  - Databricks
  - Apache Impala / Kudu
  - Apache Spark

Column oriented Databases - Parquet

- file.parquet
  - Easy to store & re-use
  - Supported by pandas Python
    - often used by data engineers & data scientists
  - No easy update (full rewrite)
- How to handle update / deletion?
  - Add a separate file with the changes
  - Delta Lake (Databricks), Apache Kudu, ...



Column oriented Databases - Parquet

- Write

```
import numpy
import pandas
import pyarrow

df = pandas.DataFrame({'one': [-1, numpy.nan, 2.5],
                       'two': ['foo', 'bar', 'baz'],
                       'three': [True, False, True]},
                      index=list('abc'))
table = pyarrow.Table.from_pandas(df)
pyarrow.parquet.write_table(table, 'example.parquet')
```

- Read

```
import pyarrow

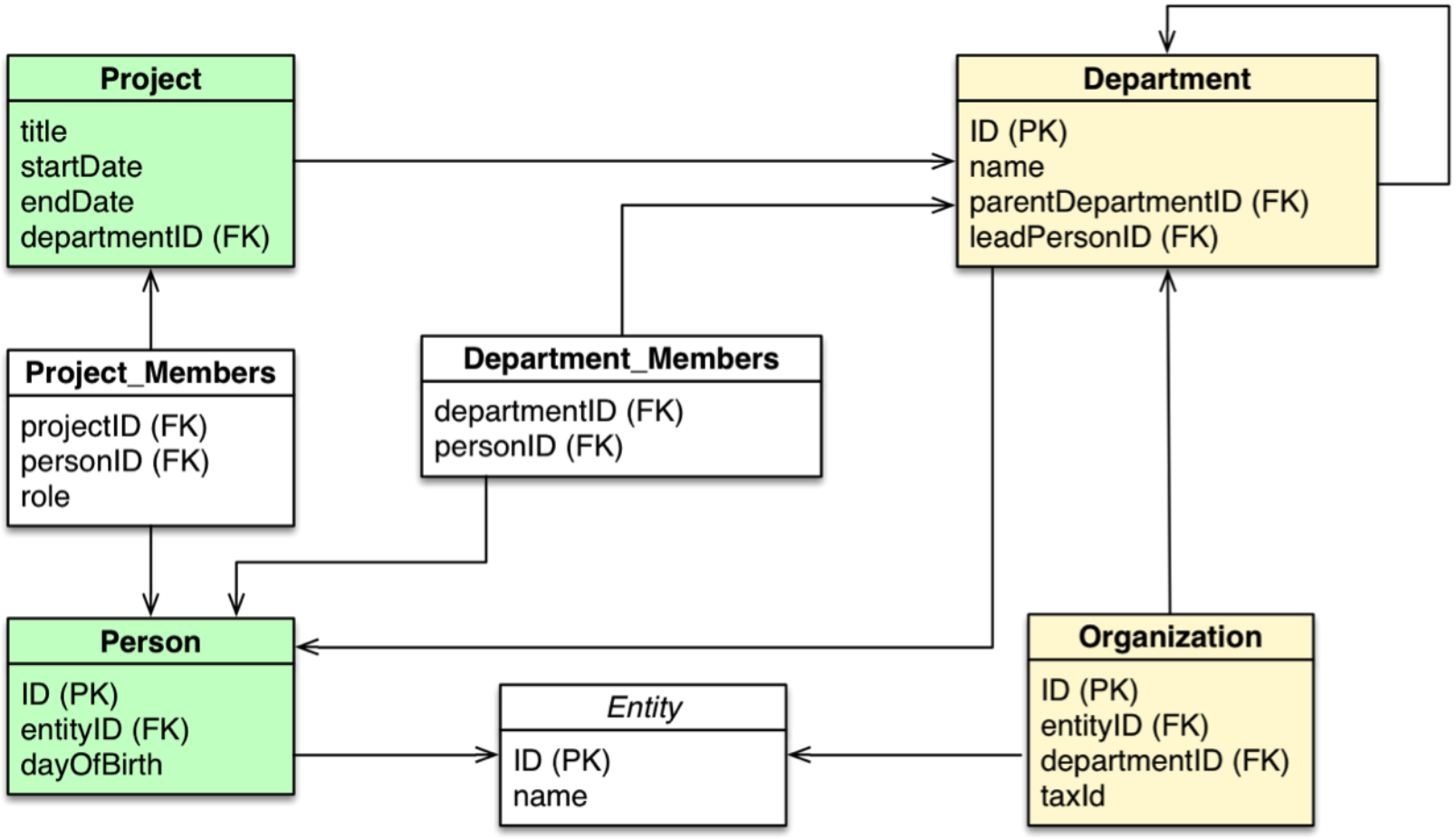
table_from_file = pyarrow.parquet.read_table('example.parquet')
```

Column oriented Databases - Delta Lake

- How to handle updates ?
  - Delta Table (open format)
- Delta Lake
  - Open source storage framework
- Python library with Rust bindings

```
from deltalake import DeltaTable, write_deltalake
import pandas as pd

# write some data into a delta table
df = pd.DataFrame({"id": [1, 2], "value": ["foo", "boo"]})
write_deltalake("./data/delta", df, mode= "append")

# Load data from the delta table
dt = DeltaTable("./data/delta")
df2 = dt.to_pandas()

assert df.equals(df2)
```



Column oriented Databases - TimescaleDB

- TimescaleDB
  - On top of Postgres, which is row oriented
  - Optimized for timeseries data
  - "Row-columnar" compression
    - Not a true column oriented database
- Benchmarks provided by TimescaleDB
  - 10% of the storage of Postgres, 350x faster queries
  - 10% of the cost of Cassandra, 5800 faster queries
  - But those are "benchmarks"

Column oriented Databases - TimescaleDB



- How do you update old records?
- What about horizontal scalability?
  - https://github.com/timescale/timescaledb/blob/main/docs/MultiNodeDeprecation.md
  - Complex, small audience

Exercise 3 - Parquet

- New features
  1. Export the data as a single parquet file having the following schema

     `order_id | order_timestamp | order_price | object_name | user_id | user_name`
  2. Each time a new order is created, you must add the order to the parquet table
     - This should be executed in less than 1s regardless of the total parquet file size
  3. Add a feature to delete an arbitrary order (by providing its order_id) from the database, and automatically remove it from the parquet file table
     - This should be executed in less than 1s regardless of the total parquet file size
  4. With 50 millions orders, 10 users, 10 objects
     1. How much time does it take to load all (order_id, order_price) from sqlite?
     2. And how much from your parquet file?

Document oriented Databases

- JSON are everywhere anyway
  - Store directly the data as you want to display it
  - Easy to add information
- How do you design a data model with plugins bringing their own state?
  - New columns ? What about their default values ?
  - How will it work for the core process?
    `INSERT INTO table VALUES (?, ?)`
- Magical solution to all your problems?
  - Data duplication
  - Deletion of an embedded entity
  - Documents can have many different schemas

```
{
  "id": 0,
  "name": "Alex",
  "classes": [
    {
      "id": 42,
      "name": "Programming"
    }
  ],
  "scores": [
    {
      "test": {
        "id": 25,
        "name": "1st test"
      },
      "score": 20
    }
  ]
}
```

Document oriented Databases - MongoDB

- Alternative by Amazon AWS: DocumentDB
- Tables are called Collections - No need to create them!
- You can index any field, even within arrays
- Custom query language
  - Filtering
  - Advanced transformations
- Designed for scalability and redundancy
- Shell is javascript-ish
- Each document < 16MB
  - Unless GridFS
- Users are handled by database

Document oriented Databases - MongoDB

- Insert data

```
db.sales.insertMany( [
   {
      _id: 0,
      items: [
         { item_id: 43, quantity: 2, price: 10, name: "pen" },
         { item_id: 2, quantity: 1, price: 240, name: "briefcase" }
      ]
   },
   {
      _id: 1,
      items: [
         { item_id: 23, quantity: 3, price: 110, name: "notebook" },
         { item_id: 103, quantity: 4, price: 5, name: "pen" },
         { item_id: 38, quantity: 1, price: 300, name: "printer" }
      ]
   }
])
```

Document oriented Databases - MongoDB

- Filter data

```
db.sales.aggregate( [
   {
      $project: {
         items: {
            $filter: {
               input: "$items",
               as: "item",
               cond: { $eq: [ "$$item.name", "pen"] }
            }
         }
      }
   }
] )
```

```
[
    {
      _id: 0,
      items: [ { item_id: 43, quantity: 2, price: 10, name: 'pen' } ]
   },
   {
      _id: 1,
      items: [ { item_id: 103, quantity: 4, price: 5, name: 'pen' } ]
   },
   { _id: 2, items: [] }
]
```

Document oriented Databases - MongoDB

- Aggregate stages
  - $bucket
  - $count
  - $geoNear
  - $group
  - $limit
  - $match
  - $lookup
  - …
- What about JOINS
  - $lookup is historically not-efficient, and not designed for sharded collections (better since 5.1)
- Some other interesting features
  - Time to live indexes
  - Capped collections

Document oriented Databases - MongoDB

- How to avoid id collisions with NoSQL Databases?
  - Should we use auto-incremental id?
- PostgreSQL (and others)
  - UUID: `123e4567-e89b-12d3-a456-426614174000`
- MongoDB
  - ObjectID: `507f1f77bcf86cd799439011`
  - Not a string!
  - Not mandatory
- Other NoSQL databases
  - Most of the time, up to you



Document oriented Databases - CouchDB

- Similarities with MongoDB
  - Document oriented with JSON-based document
- Differences
  - Availability over Consistency
  - Eventually consistent
  - "Optimistic Concurrency"
  - ! No lock on writes
  - HTTP Rest API
  - (Very) Limited search features
    - Basic find & get
    - Must use Views & Map-Reduce for advanced queries
  - No transaction

Document oriented Databases - CouchDB

- CLI
  - curl -X PUT http://127.0.0.1:5984/mydb/"id" -d ' { document} '
    - Always provide the full document, no partial update possible
  - curl -X GET http://127.0.0.1:5984/mydb/**001**
  - `curl -X DELETE http://127.0.0.1:5984/mydb/id?rev=rev_number`
    - `curl -X GET http://127.0.0.1:5984/my_database/001`
    - `{"_id":"001","_rev":"2-04d8eac1680d237ca25b68b36b8899d3","age":"23" }`
    - `curl -X DELETE http://127.0.0.1:5984/my_database/001?rev=2-04d8eac1680d237ca25b68b36b8899d3`
    - `curl -vX PUT http://127.0.0.1:5984/database_name/database_id /filename?rev=document rev_id --data-binary @filename -H "Content-Type: type of the content"`
    - curl -X PUT http://admin:password@127.0.0.1:5984/db/_design/my_ddoc -d '{"views":{"my_filter":{"map": "function(doc) { if(doc.date && doc.title) { emit(doc.date, doc.title); }}"}}}'

Document oriented Databases - CouchDB

- CLI
  - curl -X PUT http://127.0.0.1:5984/mydb/"id" -d ' { document} '
    - Always provide the full document, no partial update possible
  - curl -X GET http://127.0.0.1:5984/mydb/**001**
  - `curl -X DELETE http://127.0.0.1:5984/mydb/id?rev=rev_number`
    - `curl -X GET http://127.0.0.1:5984/my_database/001`
    - `{"_id":"001","_rev":"2-04d8eac1680d237ca25b68b36b8899d3","age":"23" }`
    - `curl -X DELETE http://127.0.0.1:5984/my_database/001?rev=2-04d8eac1680d237ca25b68b36b8899d3`
    - `curl -vX PUT http://127.0.0.1:5984/database_name/database_id /filename?rev=document rev_id --data-binary @filename -H "Content-Type: type of the content"`
    - curl -X PUT http://admin:password@127.0.0.1:5984/db/_design/my_ddoc -d '{"views":{"my_filter":{"map": "function(doc) { if(doc.date && doc.title) { emit(doc.date, doc.title); }}"}}}'
- Query server
  - Execute Javascript code for views

Document oriented Databases - CouchDB

- Views in a SQL database
  - no major differences with NoSQL databases, concept is the same



Exercise 4 - MongoDB

- New features
  1. When writing to sqlite, also write the data to a MongoDB database in separate collections
  2. Create an optimized collection to know everything about an arbitrary User, and display the result for a user x
  3. Add a feature to delete an arbitrary shopping object
     1. Are the related collections consistent? Is it useful/problematic for the future ?

Exercise 5 - MongoDB

- New features
  1. (Try to) insert 2 users, 10 objects and 50 000 orders for both users. Fix your application.
     1. Note: If there is no issue at first, increase the number of orders until there is one
  2. Make sure orders created 2 minutes ago are automatically deleted from the database wherever possible, using as few lines of code as possible
  3. Add a feature to rename an object everywhere
  4. Add a feature to change the price of an object starting from now

Graph oriented Databases

- Facebook experiment
  - Six degrees of separation (play): Everybody on this planet is separated by only six other people
  - Facebook friend graph for 721 millions users (2011): 3.74
  - Facebook friend graph for 1.59 billions users (2016): 3.57
  - https://research.facebook.com/blog/2016/2/three-and-a-half-degrees-of-separation/
- How to do it with a SQL Database?
  - user1_id | user2_id
  - Average of 200 friends / user: ~1.5 billions * 200
- How to do it with a NoSQL Database?
  - Key/Value ?
  - Column oriented?
  - Document?

Graph oriented Databases

- Graph structure for queries
  - Nodes (entities)
  - Edges (relationships)
  - Properties (information associated to node)
  - Labels
- Graph databases
  - Neo4j
  - OrientDB
  - ArangoDB
  - Amazon Neptune
- Plugins for non-graph databases
  - MongoDB
  - Apache AGE (on top of Postgres)

Graph oriented Databases



Graph oriented Databases - Neo4j

- Neo4j



  - Cypher
  - `(emil)<-[:KNOWS]-(jim)-[:KNOWS]->(ian)-[:KNOWS]->(emil)`
  - ```
    (emil:Person {name:'Emil'})
      <-[:KNOWS]-(jim:Person {name:'Jim'})-[:KNOWS]->
      (ian:Person {name:'Ian'}) -[:KNOWS]->(emil)
    ```
    ```

Graph oriented Databases - Neo4j



Graph oriented Databases - Neo4j

- Employees of the IT department ?
  - ```
    SELECT name FROM Person
    LEFT JOIN Person_Department ON Person.Id = Person_Department.PersonId
    LEFT JOIN Department ON Department.Id = Person_Department.DepartmentId
    WHERE Department.name = "IT Department"
    ```
  - ```
    MATCH (p:Person)<-[:EMPLOYEE]-(d:Department)
    WHERE d.name = "IT Department"
    RETURN p.name
    ```
- For each customer who bought a product, look at the products that peer customers have purchased and add them as recommendations. Discard already purchased items.
  - ```
    MATCH (u:Customer {customer_id:'customer-one'})-[:BOUGHT]->(p:Product)<-
    [:BOUGHT]-(peer:Customer)-[:BOUGHT]->(reco:Product)
    WHERE not (u)-[:BOUGHT]->(reco)
    RETURN reco as Recommendation, count(*) as Frequency ORDER BY Frequency DESC
    LIMIT 5;
    ```
    ```

Graph oriented Databases - Neo4j

- Equivalent SQL
  - ```
    SELECT product.product_name as Recommendation, count(1) as Frequency FROM
    product, customer_product_mapping,
     (SELECT cpm3.product_id, cpm3.customer_id
        FROM Customer_product_mapping cpm, Customer_product_mapping cpm2,
    Customer_product_mapping cpm3
        WHERE cpm.customer_id = 'customer-one' and cpm.product_id = cpm2.product_id
    and cpm2.customer_id != 'customer-one' and cpm3.customer_id = cpm2.customer_id
    and cpm3.product_id not in (SELECT distinct product_id FROM
    Customer_product_mapping cpm WHERE cpm.customer_id = 'customer-one')
     ) recommended_products
    WHERE customer_product_mapping.product_id = product.product_id and
    customer_product_mapping.product_id in recommended_products.product_id and
    customer_product_mapping.customer_id = recommended_products.customer_id
    GROUP BY product.product_name
    ORDER BY Frequency desc
    ```
    ```

Graph oriented Databases - Neo4j

- CLI
  - `CREATE (node1),(node2)`
  - `CREATE (node:label)`
  - `CREATE (node1)-[:RelationshipType]->(node2)`
    - `CREATE (Alex:player{name: "Alex Paq", birth_year: 1990})`
      `CREATE (BE:Country {name: "Belgium"})`
    - `CREATE (Alex)-[r:LIVE_IN]->(BE)`
  - `MATCH (a:player), (b:Country) WHERE a.name = "Alex Paq" AND b.name = "Belgium"`
    `CREATE (a)-[r: LIVE_IN]->(b)`
    `RETURN a,b`
  - `CREATE (node1)-[label:Rel_Type {key1:value1, key2:value2, . . . n}]->(node2)`
  - `MERGE (node:label) RETURN node`

Graph oriented Databases - Neo4j

- CLI
  - `MATCH (node:label)`
    `RETURN node`
  - `MATCH (node:label)<-[: Relationship]-(n)`
    `RETURN n`
    - `MATCH (BE:Country {name: "Belgium", result: "Winners"})<-[: TOP_SCORER_OF]-(n)`
      `RETURN n.name`
  - `MATCH (label) WHERE label.country = "property"`
    `RETURN label`
  - `MATCH (n) DETACH DELETE n`
  - `MATCH (Alex:player {name: "Alex Paq", birth_year: 1990}) DETACH DELETE Ishant`
  - `MATCH p = (start node)-[*]->(end node)`
    `WHERE start.node = "node_name" AND end.node = "node_name"`
    `FOREACH (n IN nodes(p)| SET n.marked = TRUE)`

Graph oriented Databases

- When to use them?
  - Graph algorithms
  - Shortest path
  - Breadth first search
  - Depth first search
  - …
- Is it more efficient than a SQL Database or non-graph database?
  - https://neo4j.com/news/how-much-faster-is-a-graph-database-really/
  - Depth = 2 -> Same speed
  - Depth = 4 -> 1100x faster
  - Well, there are other things than finding friends…

Exercise 5 - Neo4j

- New features
    1. Create a "fork" of your project to only use Neo4j as database backend instead of SQLite / MongoDB
        1. You can keep Redis for the caching layer

Exercise 6 - Neo4j

- New features
  1. Introduce the concept of "friends" in your application: they share similar interest in the products they might be purchasing
  2. Create randomly 10 000 users, each user having between 5 & 500 friends
  3. From an arbitrary user x, compute on demand the degree of separation for every other user + the average of it
  4. Do the same computation with the original SQL database (before exercise 4)
  5. Make sure the degree of separation can be shown quickly for any user (< 10ms)