MET CS 622: File I/O and Databases

Reza Rawassizadeh

Why File and Databases?

- So far we have learned that we can keep data in a collection and files.

- Collections are very useful, but they keep the data in the memory and not on the disk.

- Memory is temporary storage for data, and after the power shuts off or is disconnected, its content will fade away.

- Therefore, we need a long-term place to keep our data, and storing data in a file is a solution.

Why File and Databases?

- Files are very useful permanent storage of data, and they keep the data on the disk.

- However, when the data gets large it is a cumbersome process to find a piece of information from the file.

- In particular, querying information from a raw file is not easy. Not easy here means not resource efficient (time, energy,…)

- To mitigate this challenge databases have been introduced.

Outline

- File
- Databases
- Introduction to DBs
- Query Languages
- SQL Database
- NoSQL Database

Outline

- **File**
- Databases
  - Introduction to DBs
  - Query Languages
  - SQL Database
  - NoSQL Database

File Concepts

- Directory
- Root Directory
- File
- Path (Absolute, Relative)
- File Separator

```
System.out.println(System.getProperty("file.separator"));
System.out.println(java.io.File.separator);
```

Sample Code to Check File Existence

```
import java.io.File;

public class FileSample {
public static void main(String[] args) {
  File file = new File(“/home/foo/tessst.txt”);
   System.out.println(file.exists());
   }
}
```

File Object

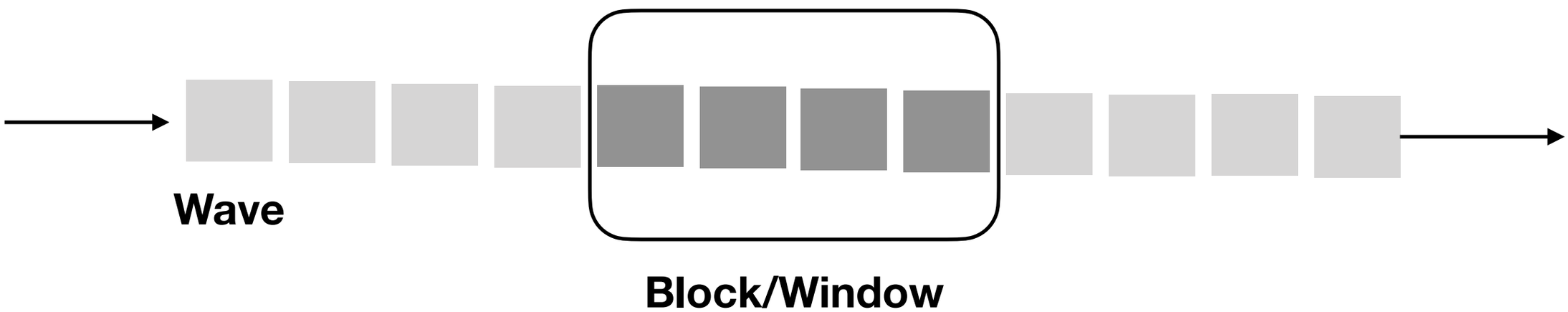
- `exists()`
- `getName()`
- `getAbsolutePath()`
- `isDirectory()`
- `isFile()`
- `length()`
- `lastModified()`
- `delete()`
- `renameTo(File)`
- `mkdir()`
- `getParent()`
- `listFiles()`

File Example
(do it in the class)

```
import java.io.File;
public class ReadFileInformation {
 public static void main(String[] args) {
  File file = new File(“C:\\data\\ttttest.txt”);
  System.out.println("File Exists: “+file.exists());
  if(file.exists()) {
    System.out.println("Absolute Path: “+file.getAbsolutePath());
    System.out.println("Is Directory: “+file.isDirectory());
    System.out.println("Parent Path: "+file.getParent());
    if(file.isFile()) {
      System.out.println("File size: "+file.length());
      System.out.println("File LastModified: "+file.lastModified()); }
    else {
      for(File subfile: file.listFiles())}
       { System.out.println("\t"+subfile.getName());
    } // 2nd if
   } // 1st if
 } // main
} // class
```

Stream

- Stream is a never ending dataset.
- For example, running application log files, online financial transactions, sensor data,…
- Stream processing allow the application to access only small portion of data.



`java.io`

- `java.io` defines two set of classes to work with file.

1. Classes with `Stream` in their name, e.g. `FileInputStream`.

2. Classes with `Reader/Writer` in their name, e.g. `FileReader`. They are still dealing with streams but the name stream is not mentioned in their name.

- Stream classes are used to read/write any binary files, (e.g. .png, .avi, .mp3) but reader/writer classes are used to read and write character files (e.g. .txt)

`java.io`

- For most “input streams” there is a similar “output stream” existed, e.g. `FileInputStream` and `FileOutputStream`.

- Also, most “reader” classes have corresponding “writer” classes as well. e.g. `FileWriter` and `FileReader`.
  But PrintWriter has no PrintReader.

- Streams are categorized into low-level streams and high-level streams.

BaseClasses

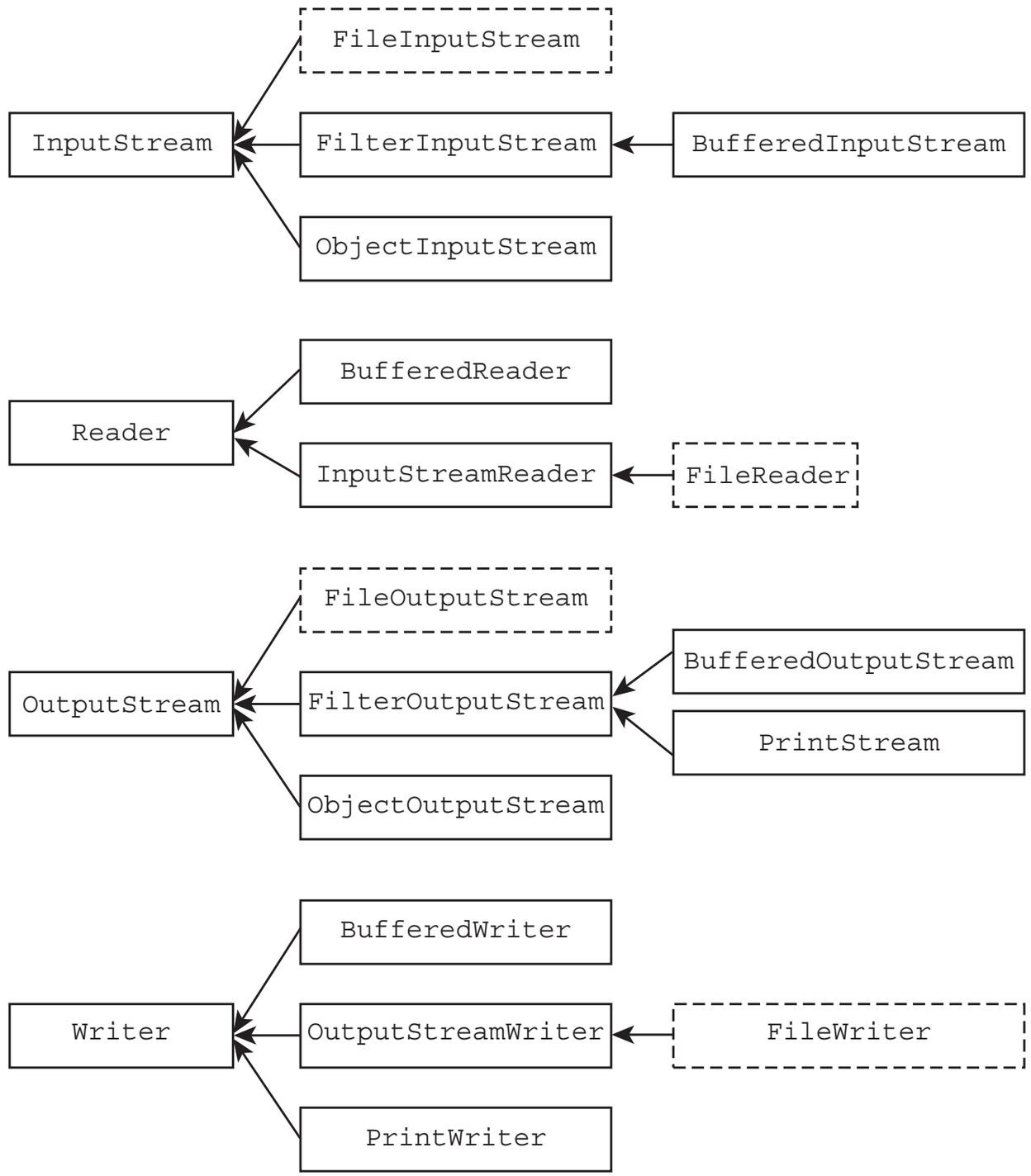
- `java.io` defines four bases classes that constitute the foundation of java file operations, including: `InputStream`, `OutputStream`, `Reader` and `Writer`.

- Children classes of these classes include their name, e.g. `ObjectOutputStream, FileReader, …`

Review of `java.io` Class Properties

- A class with the word InputStream or OutputStream in its name is used for *reading or writing binary data*, respectively.
- A class with the word Reader or Writer in its name is used for reading or writing *character or string data*, respectively.
- Most, but not all, input classes have a corresponding output class.
- A low-level stream connects directly with the source of the data.
- A high-level stream is built on top of another stream using wrapping.
- A class with *Buffered* in its name reads or writes data in groups of bytes or characters and often *improves performance* in sequential file systems.

java.io classes



Low level and High level Streams

- Low level streams connects directly to the file on the disk.

- A *high-level stream* is built on top of another stream using wrapping. Wrapping is the process that passes an instance of a class to a constructor of another class.

high level

low level

```
try {
  BufferedReader bufferedReader = new BufferedReader(
    new FileReader(“test.txt"))) {
      System.out.println(bufferedReader.readLine());
}
```

Buffered Reader

- Buffered classes are reading or writing data in a chunk and not byte per byte or character per character. Therefore, disk access operations are getting less and this leads to a significant performance improvement.

- Do not forget to use Buffered reader, especially in your coding exams. Exam takers are usually sensitive about this.

‘Not’ Compiling Examples

```
new BufferedInputStream(new FileReader("test.txt"));

new BufferedWriter(new FileOutputStream("ttt.txt"));
```

```
new ObjectInputStream(new FileOutputStream(“ttt.txt"));

new BufferedInputStream(new InputStream());
```

Serialization and Deserialization

- The process of **converting an in-memory object to a stored data format is referred to as *serialization***, with the reciprocal process of **converting stored data into an object, which is known as *deserialization***.

- To make an object serializable that class should implement `java.io.Serializable`.

- Many of the java objects we use like String, implements serializable.

- Note in a hierarchy of objects all of them must be marked serializable. For example, if we mark *student* object serializable its *courses* which is another object inside the student object should be serializable as well.

Making an Example Class Serializable

```
import java.io.Serializable;
public class Animal implements Serializable {
 private static final long serialVersionUID = 1L;
 private String name;
 private int age;
 private char type;
public Animal(String name, int age, char type) {
 this.name = name;
 this.age = age;
 this.type = type;
}
public String getName() { return name; }
public int getAge()  { return age; }
public char getType() { return type; }
public String toString() {
  return "Animal [name=" + name + ", age=" + age + ", type=" + type + "]";
}}
```

```
FileOutputStream fout = new
FileOutputStream(“..path..”);
ObjectOutputStream oos =
newObjectOutputStream(fout);
oos.writeObject(Animal);
```

Making an Example Class Serializable

```
import java.io.Serializable;
public class Animal implements Serializable {
 private static final  long serialVersionUID = 1L;
 private String name;
 private int age;
 private char type;
 public Animal(Stri
  this.name = name;
  this.age = age;
  this.type = type;
}
public String getN
public int getAge()  { return age; }
public char getType() { return type; }
public String toString() {
  return "Animal [name=" + name + ", age=" + age + ", type=" + type + "]";
}}
```

It is not mandatory to have serialVersionUID, but it is recommended to do so.
Otherwise, java is going to creates it for us.

Java I/O Example 1
Creating a file

```
import java.io.*;
public class FileExample {
    public static void main(String[] args) {
        try {
            File file = new File("javaFile123.txt");
            if (file.createNewFile()) {
                System.out.println("New File is created!");
            } else {
                System.out.println("File already exists.");
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Java I/O Example 2
List content of a folder

```
import java.io.*;
public class FileExample {
   public static void main(String[] args) {
      File f = new File(“/Users/home/foo/“);
      String filenames[] = f.list();
      for(String filename:filenames){
          System.out.println(filename);
      }
   }
}
```

Java I/O Example 3
Delete a file

```
import java.io.File;

public class MyMain {
   public static void main(String[] args) throws Exception {
      deleteDir(new File("/home/xxxxx"));
   }
 public static boolean deleteDir(File dir) {
    if (dir.isDirectory()) {
       String[] children = dir.list();
       for (int i = 0; i < children.length; i++) {
          boolean success = deleteDir(new File(dir, children[i]));
          if (!success) {
             return false;
          }
       }
    }
    return dir.delete();
}
```

Java I/O Example 4
Read the content of a text file

```
package edu.met622.testio;
import java.io.*;
public class MyFileReader {
    public static void main(String[] args) {
        String testFile = “/Users/home/test/test.txt”;
        BufferedReader br = null;
        String line = "";
        try {
            br = new BufferedReader(new FileReader(testFile));
          while ((line = br.readLine()) != null) {
              System.out.println(“read:” +line);
          }
      } catch (FileNotFoundException e) {
          e.printStackTrace();
      } catch (IOException e) {
          e.printStackTrace();
      } finally {
          if (br != null) {
              try {
                  br.close();
              } catch (IOException e) {
                  e.prinatStackTrace();
              }
          }}}
```

Java I/O Example 5
Copy binary Files

```
import java.io.*;
public class CopyBufferFileSample {
  public static void copy(File source, File dest) throws  IOException {
  try {
          FileInputStream fin = new FileInputStream(source);
          FileOutputStream fout = new FileOutputStream(dest);
          byte[] b = new byte[1024];
          int noOfBytes = 0;
          System.out.println("Copying file using streams");
             while( (noOfBytes = fin.read(b)) != -1 ){
              fout.write(b, 0, noOfBytes);
          }
          System.out.println("File copied!");
          //close both streams
          fin.close();
          fout.close();
      }
      catch(FileNotFoundException fnf){
          System.out.println("Specified file not found :" + fnf);
      }
      catch(IOException ioe){
          System.out.println("Error while copying file :" + ioe);
      }
    }
}
```

Example 6
Read from file

- Create a mock text (.txt) file with code (not by hand).
- Write inside that file for 30 times following sentences:

  I am expert in working with java I/O.

Copy a text file

```
import java.io.*; import java.util.*;
public class CopyTextFileSample {
public static List<String> readFile(File source) throws IOException {
List<String> data = new ArrayList<String>();
try (BufferedReader reader = new BufferedReader(new FileReader(source))) {
String s;
while((s = reader.readLine()) != null) {
data.add(s); }
}
return data; }
```

```
public static void writeFile(List<String> data, File destination) throws IOException
{
try (BufferedWriter writer = new BufferedWriter(
new FileWriter(destination))) {
for(String s: data) { writer.write(s);
writer.newLine(); }
} }
```

```
public static void main(String[] args) throws IOException { File source = new
File("Zoo.csv");
File destination = new File("ZooCopy.csv");
List<String> data = readFile(source);
for(String record: data) { System.out.println(record);
}
writeFile(data,destination); }
```

Outline

- File
- **Databases**
- **Introduction to DBs**
- Query Languages
- SQL Database
- NoSQL Database

Why Database?

- File storages are the first version of digital data preservation.
- Nevertheless, they are rapidly getting very large and thus searching for an information is getting complicated and resource consuming.
- Therefore, developers and research communities identify the need for an optimal query languages, and thus they introduced databases.

Common Data Storage Formats

- CSV, TSV
- XML
- JSON
- Relational Databases (SQL)
- Graph Formats (SPARQL, Germlin,..)

XML and JSON

Tabular data storages are not good for dealing with hierarchy of data. To handle this situation XML and JSON could be used.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Example>
    <Course id="met622">
        <!-- here I can write my comment-->
        <name>Advanced Programming Techniques</name>
        <participants>
            <student id='1'/>
            <student id='2' />
            <student id='10' />
        </participants>
    </Course>
</Example>
```

```json
{
   "Example": {
      "Course": {
         "name": "Advanced Programming Techniques",
         "participants": {
            "student": [{"id":"1"},{"_id":"2"},{"_id":"10"}]
         },
         "id": "met622"
      }
   }
}
```

- JSON is more compact than XML.
- JSON is easier to use with JScript.
- XML has support for comments, but JSON has not support for comments.
- XML has XSLT to transform XML document into different format.
- JSON is easier to use for arrays, lists and data collections.

Database

- Database and modeling how to deal with the data is the most important part of software engineering because it describes our interpretation of the problem.

- There are types of databases, **relational databases** (SQL) and **document databases**, **hierarchical databases,** and **graph databases**.

- The roots of **relational** databases lie in **business data** processing, especially for their **transaction processing** and **batch processing** capabilities.

- There are lots of competitors against Relational Databases (RDB) including XML databases, Object relational databases, etc. None of them lasted for a significant time and until now RDB is the most used DB.

- Today, still relational databases have an important role. However, the advent of big data emerges document databases as well. Therefore, in enterprise software, we will encounter both types of databases.

History of Databases

- The early version of databases is document models.

- SQL which has introduced in 1970 by Edgar Codd, is still the widest used language to query information and leads to the introduction of relational databases. The root of relational data processing is going back to business data processing, in 60's and 70’s.

- In early 2000, object-relational databases (ORB) get some attention, and a bit more flexible formats such as XML databases were introduced. They were not successful against relational databases and didn’t survive.

- One big challenge with ORB is that they can not provide an accurate matching between database and application layer.

- NoSQL is the most recent attempt to improve database query languages. It has been derived in 2009, with a catchy Twitter hashtag.

History of Databases

- SQL has some limitations including (i) **the need for schema**, (ii) **performance challenges** while dealing with very large data, (iii) commercialization of SQL databases and their **high cost**, (iv) **lack of dynamic data models**.

- NoSQL databases focus on mitigating these issues, and nowadays they have a significant share of the market.

- Still, relational databases and their query language (SQL) have the highest contribution in the market of databases.

Databases

- Hierarchical Models
- Relational Databases (SQL)
- Document Databases (NoSQL)
- Graph Databases

Databases

- **Hierarchical Models**
- Relational Databases (SQL)
- Document Databases (NoSQL)
- Graph Databases

Hierarchical Database

- The earliest implementation of databases are focused on hierarchical databases, e.g. IBM Information Management System (IMS).

- They are being used widely prior to 70's.

- Hierarchical databases work well for one-to-many relationships, but are not easy for many-to-many relationships.

- They do not support joins and their limitation leads to the introduction of relational databases in 1970

Databases

- Hierarchical Models
- **Relational Databases (SQL)**
- Document Databases (NoSQL)
- Graph Databases

Relational Databases

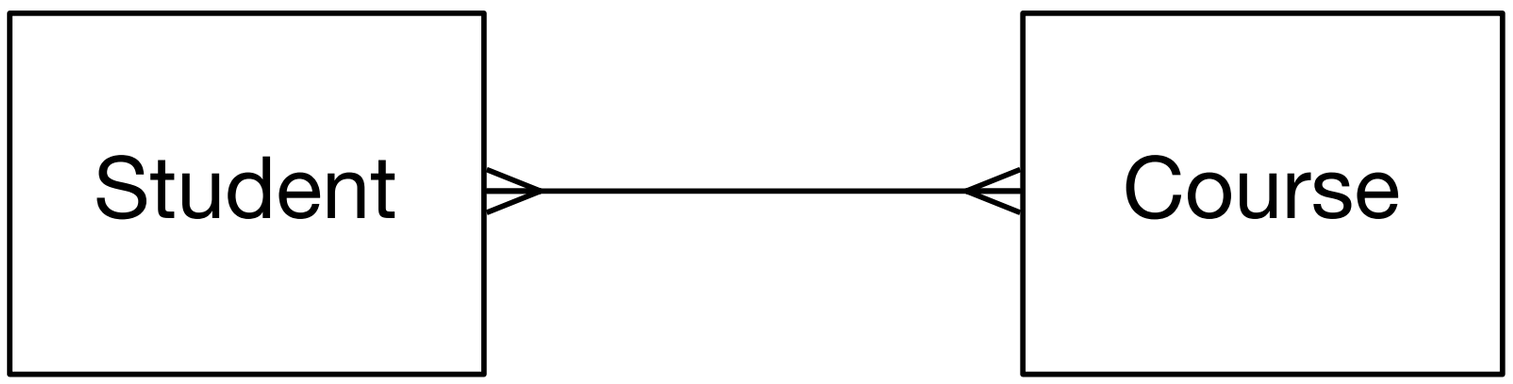
- They are operated optimal when we have a Many-to-One relationship.

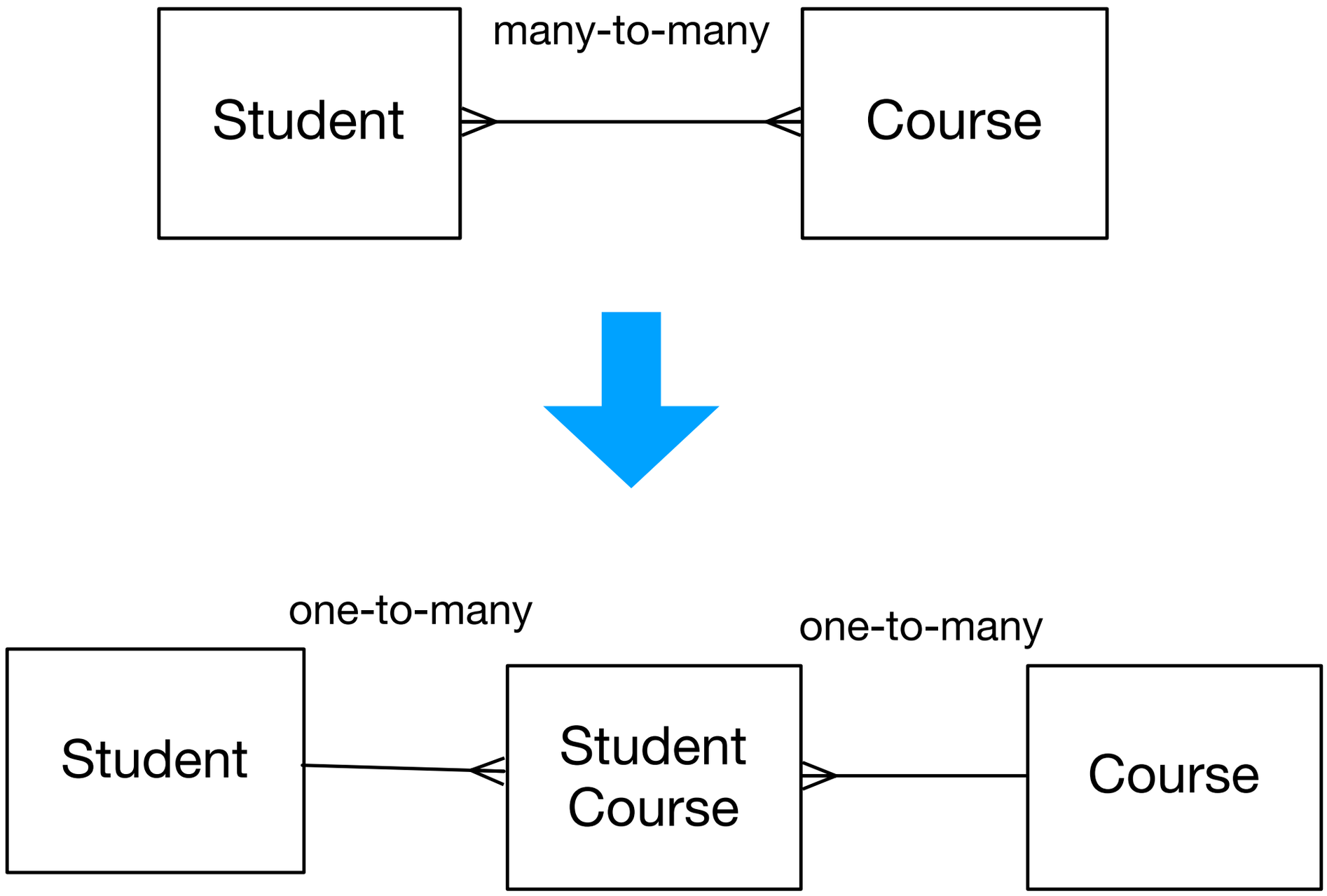
- The introduction of relational databases and their query language, SQL, is still one of the most used features in software engineering community.

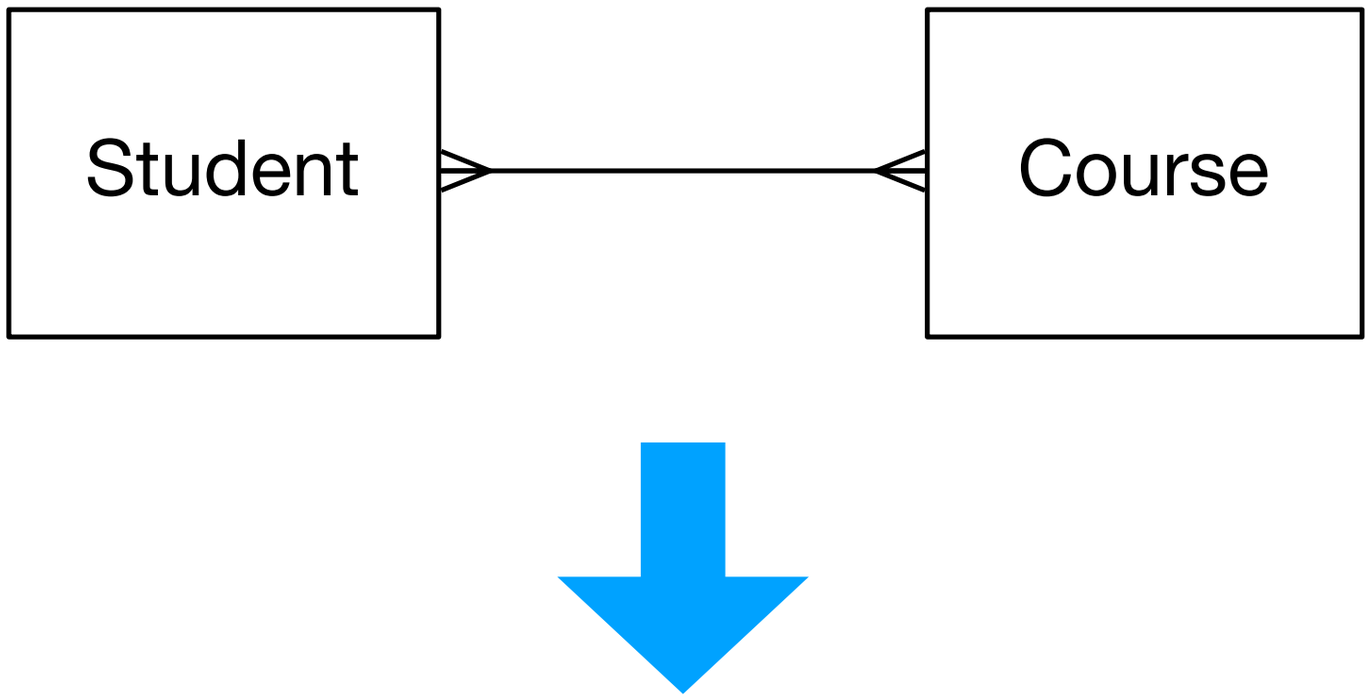
- To handle a Many-to-Many relation we need to normalize that relation into One-to-Many relations.

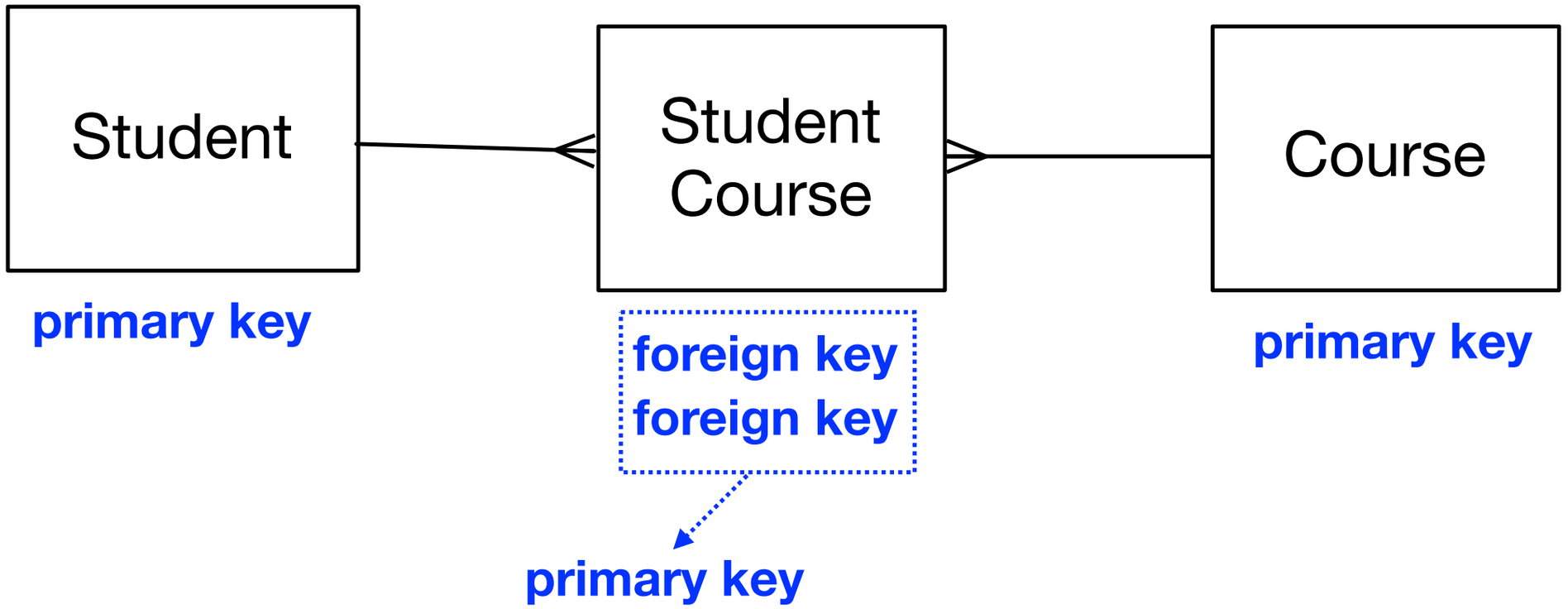
Relational Database Normalization

The main objective of normalization in relational databases is to convert **Many-to-Many** to **One-to-Many** relations.

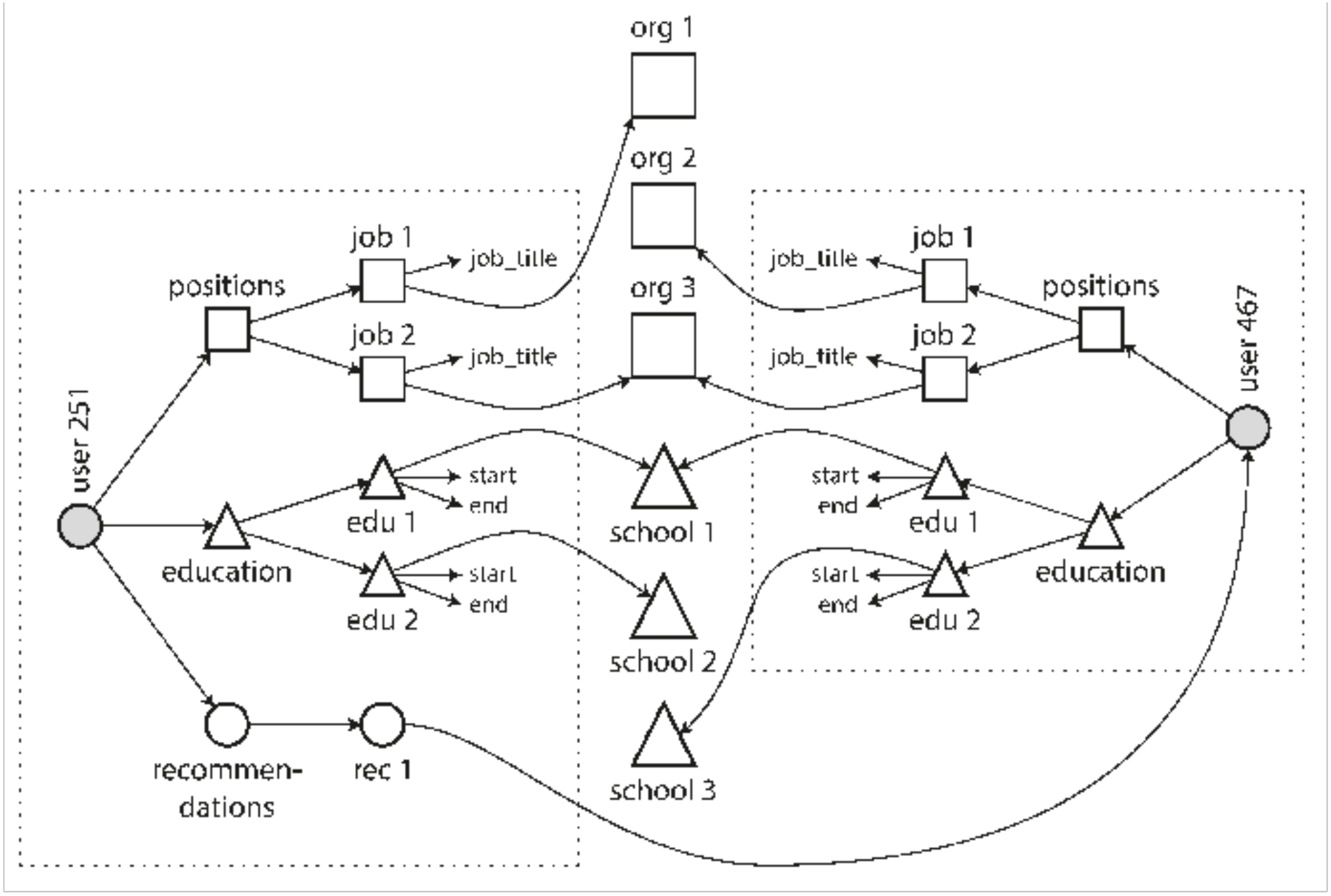



Student
many-to-many
Course
one-to-many
Student
Student
Course
one-to-many
Course


Student
Course


Student
Student Course
Course
primary key
foreign key
foreign key
primary key
primary key

Some times normalization makes the database very complicated



Source: Designing Data-Intensive Applications book

Databases

- Hierarchical Models
- Relational Databases
- **Document Databases (NoSQL)**
- Graph Databases

Document Databases

- NoSQL databases as a buzzword are emerged in 2010s and they are the new competitor against relational databases. Nevertheless, each has their own share in the community.

- Document databases address following weaknesses existed in traditional relational databases (RDB):

- A need for greater scalability, especially for very large datasets which require very high throughput.

- A widespread of free and open-source software among commercialized and expensive RDBs.

- A flexible style of storing data with more freedom that doesn't exist in RDBs. In a more technical sense, document databases are schema-less. Therefore, this could be an advantage.

Document Databases

- NoSQL databases as a buzzword are emerged in 2010s and they are the new competitor against relational databases. Nevertheless, each has their own part in the community.
- Document databa in traditional relational databas
- A need for gre asets which require very hi
- A widespread ommercialized and expensive
- A flexible style of storing data with more freedom, that doesn't existed in RDBs. In more technical sense, document databases are schema-less. Therefore, this could be an advantage.

Can you give me an example about dis-advantages of having Schema ?

Example Challenge of Relational Database

http://www.linkedin.com/in/williamhgates

Bill Gates

Greater Seattle Area | Philanthropy

Summary

Co-chair of the Bill & Melinda Gates Foundation. Chairman, Microsoft Corporation. Voracious reader. Avid traveler. Active blogger.

Experience

Co-chair • Bill & Melinda Gates Foundation
2000 – Present

Co-founder, Chairman • Microsoft
1975 – Present

Education

Harvard University
1973 – 1975

Lakeside School, Seattle

Contact Info

Blog: thegatesnotes.com
Twitter: @BillGates

users table

user_id	first_name	last_name	summary
251	Bill	Gates	Co-chair of … blogger.

region_id	industry_id	photo_id
us:91	131	57817532

regions table

id	region_name
us:7	Greater Boston Area
us:91	Greater Seattle Area

industries table

id	industry_name
43	Financial Services
48	Construction
131	Philanthropy

positions table

id	user_id	job_title	organization
458	251	Co-chair	Bill & Melinda Gates F…
457	251	Co-founder, Chairman	Microsoft

education table

id	user_id	school_name	start	end
807	251	Harvard University	1973	1975
806	251	Lakeside School, Seattle	NULL	NULL

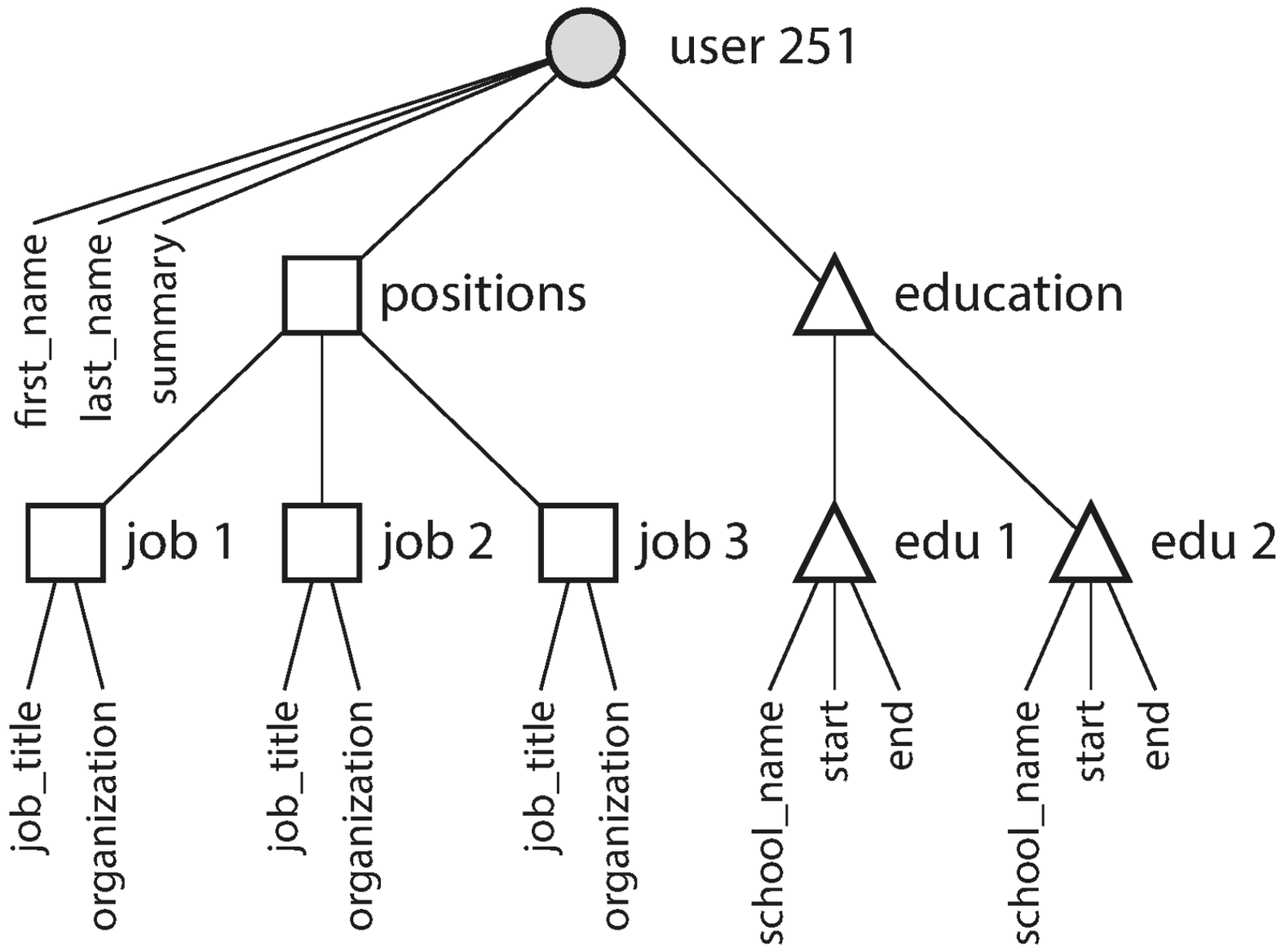
contact_info table

id	user_id	type	url
155	251	blog	http://thegatesnotes.com
156	251	twitter	http://twitter.com/BillGates

Photo of Bill Gates courtesy of Wikimedia Commons, Ricardo Stuckert, Agência Brasil.

source: Designing Data-Intensive Applications book

Query Structure of the described model



Document Database (use JSON representation) of the CV


```
{
    "user_id": 251,
    "first_name": "Bill",
    "last_name": "Gates",
    "summary": "Co-chair of the Bill & Melinda Gates... Active blogger.", "region_id":"us:91",
    "industry_id": 131,
    "photo_url": "/p/7/000/253/05b/308dd6e.jpg",
    "positions": [
        {"job_title": "Co-chair", "organization": "Bill & Melinda Gates Foundation"},
        {"job_title": "Co-founder, Chairman", "organization": "Microsoft"}
     ], "education": [
        {"school_name": "Harvard University", "start": 1973, "end": 1975},
        {"school_name": "Lakeside School, Seattle", "start": null, "end": null} ],
    "contact_info": {
        "blog": "http://thegatesnotes.com", "twitter": "http://twitter.com/BillGates"
     },
}
```

Schema Flexibility

- *Most document databases, and the JSON support in relational databases, do not enforce any schema on the data in documents. XML support in relational databases usually comes with optional schema validation. No schema means that arbitrary keys and values can be added to a document, and when reading, clients have no guarantees as to what fields the documents may contain*.
(Klepmann ’18), page 39.

Document vs Relational Databases

- Schema-less advantages in document DB, especially in altering the DB structure and updating a row in a large dataset. Schema-less is very important for heterogeneous data structures. Such as smartphone sensors data, data that a robot collects about its environment, etc.

- Relational DB can support transactions, but document DB has poor support of transactions.

- Relational DB is not as scalable as Document DB.

- Usually, to update or alter a document DB, the machine needs to load the entire database into the memory. So it is better to keep your document data small and concise as much as possible, or not implement update at all.

- Similar to hierarchical DB, document DB stores the entire data as a document. Therefore, there is no join support (there are some implementational efforts to support it).

- Relational DB provides better support for many-to-many relationships, in comparison to document DB.

Which model?
Relational or Document?

- If the data has a hierarchical structure such as a tree, it is better to use **document db**.

- If the schema of our data is dynamic and changing, it is better to use **document db**.

- If we need to segment data into a smaller unit and focus on the small number of units, **relational db** is recommended.

- If the query is complicated and requires many joins among different sources of information, perhaps (not always) a **relational db** is better.

- If the application has many-to-many relationships, **relational db** is better. Joins can be encoded inside the application code. However, they introduce additional complexity in the app as well.

- If the dataset is large and it is will stored locally on one single entity, **document db** is recommended.

Outline

- File
- Databases
- Introduction to DBs
- **Query Languages**
- SQL Database
- NoSQL Database

What are important operation in databases?

- CRUD operations are something general on all datasets.
- Create
- Read
- Update
- Delete

Query Languages

- **Declarative**: We just specify WHAT is the data we intend to retrieve and what conditions the result must met, stored, grouped or aggregated. There is no information required about HOW to retrieve the data.

- Declarative query languages hide implementation details from the query language.

- **Imperative** language provides compiler step by step details how to retrieve the data in details, e.g. DOM and SAX for parsing an XML document. In other words, the only way to access data is to start from the root and go inside the hierarchy of the data.

- Imperative codes are hard to paralyze across multiple CPUs.

Query Languages

- **Declarative**: We just specify WHAT is the data we intend to retrieve and what conditions the result must met, stored, grouped or aggre out HOW to retrieve t
- Declarat details from the query la
- **Imperat** ep details how to retriev r parsing an XML doc cess data is to start fro e data.
- Imperative codes are hard to paralyze across multiple CPUs.

-Is SQL declarative or Imperative?

-Also check online, wether “StaX” is imperative or declarative?

Declarative (XSL) vs Imperative (DOM) query example

```
<ul>
<li class="selected">

<p>Sharks</p> <ul>

<li>Great White Shark</li>
<li>Tiger Shark</li>
<li>Hammerhead Shark</li>

</ul> </li>

<li> <p>Whales</p>

<ul>
<li>Blue Whale</li>
<li>Humpback Whale</li>
<li>Fin Whale</li>

</ul> </li>

</ul>
```

Imperative

```
var liElements = document.getElementsByTagName("li");
for (var i = 0; i < liElements.length; i++) {
    if (liElements[i].className === "selected") {
        var children = liElements[i].childNodes;
        for (var j = 0; j < children.length; j++) {
            var child = children[j];
            if (child.nodeType === Node.ELEMENT_NODE && child.tagName === "P") {
                child.setAttribute("style", "background-color: blue");
            }
        }
    }
}
```

Declarative (A query that changes the color)

```
<xsl:template match="li[@class='selected']/p">
    <fo:block background-color="blue">
        <xsl:apply-templates/>
    </fo:block>
</xsl:template>
```

XPath Query

```
package com.tutorialspoint.xml;
…

public class XPathParserDemo {

   public static void main(String[] args) {

      try {
         File inputFile = new File("input.txt");
         DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
         DocumentBuilder dBuilder;
         dBuilder = dbFactory.newDocumentBuilder();

         Document doc = dBuilder.parse(inputFile);
         doc.getDocumentElement().normalize();

         XPath xPath =  XPathFactory.newInstance().newXPath();

         String expression = "/class/student";
         NodeList nodeList = (NodeList) xPath.compile(expression).evaluate(
            doc, XPathConstants.NODESET);

         for (int i = 0; i < nodeList.getLength(); i++) {
            Node nNode = nodeList.item(i);
            System.out.println("\nCurrent Element :" + nNode.getNodeName());
```

Source: https://www.tutorialspoint.com/java_xml/java_xpath_parse_document.htm

XPath Query

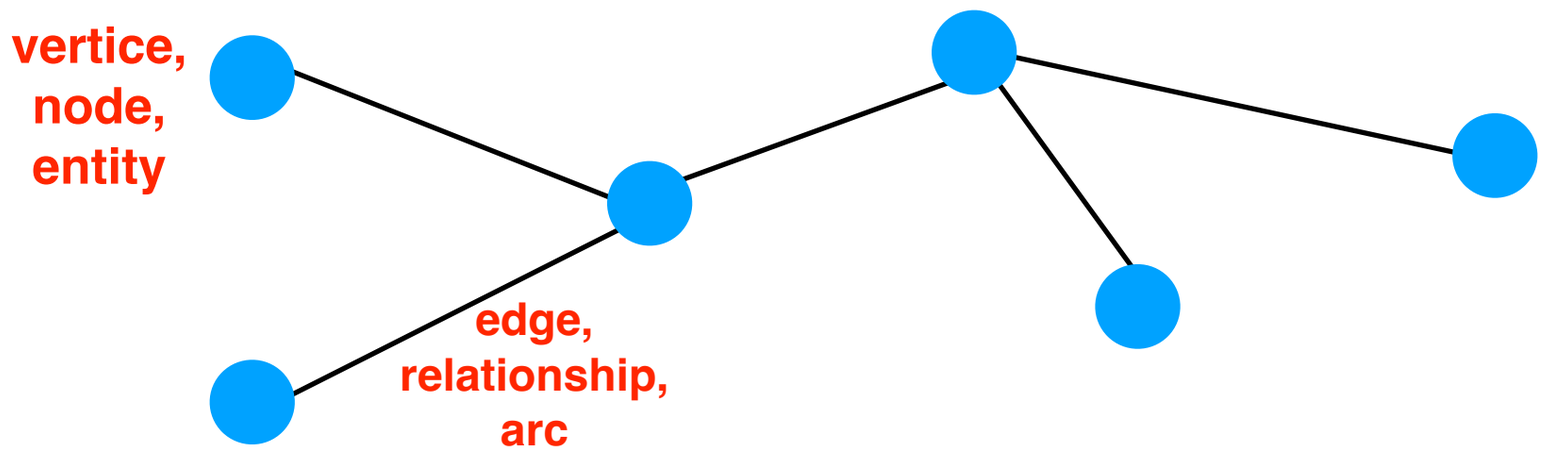
```
        String expression = "/class/student";
        NodeList nodeList = (NodeList) xPath.compile(expression).evaluate(
           doc, XPathConstants.NODESET);

        for (int i = 0; i < nodeList.getLength(); i++) {
           Node nNode = nodeList.item(i);
           System.out.println("\nCurrent Element :" + nNode.getNodeName());

           if (nNode.getNodeType() == Node.ELEMENT_NODE) {
              Element eElement = (Element) nNode;
              System.out.println("Student roll no :" + eElement.getAttribute("rollno"));
              System.out.println("First Name : "+ eElement
                 .getElementsByTagName("firstname")
                 .item(0)
                 .getTextContent());
              System.out.println("Last Name : " + eElement
                 .getElementsByTagName("lastname")
                 .item(0)
                 .getTextContent());
              System.out.println("Nick Name : " + eElement
                 .getElementsByTagName("nickname")
                 .item(0)
                 .getTextContent());
              System.out.println("Marks : "
                 + eElement
                 .getElementsByTagName("marks")
                 .item(0)
                 .getTextContent());
           }
        }
```

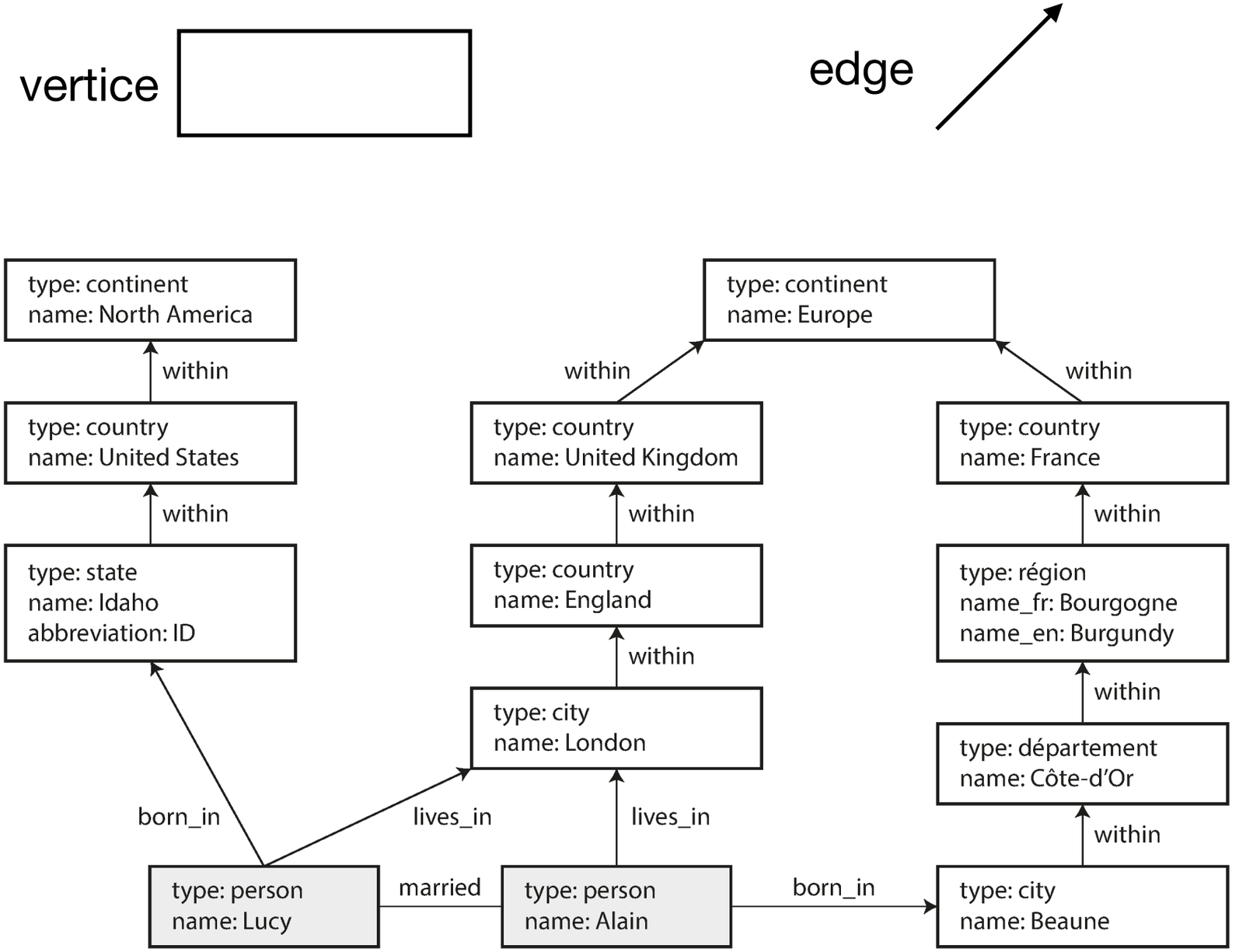
Graph Databases

- Sometimes we can not normalize many-to-many relation and it is important to distinguish feature between both entities. Both Relational and Document databases can not handle many-to-many relations well. In other words, if we need to keep many-to-many relationships between the data objects, we can use graph databases.



- Graphs are very useful to model evolving data structures, e.g. social media which connections between humans evolve. Protein structure and road network data.

Social Graph Example



This example is *Figure 2-5 of Martin Kleppmann's book*

Graph Database Examples

- **Social Graph:** A graph is used to model human connections and recommend new connections, based on mutual friends.

- **Web Pages:** The PageRank algorithm use links to a page to determine the popularity of the page.

- **Road network:** A car navigation app used graph to find the shortest path.

- **Network routing:** A routing algorithm use graph to determine to most optimal route for a message inside a network.

- **Protein Structure**

Property Graph Model

Each vertex consists of:

- A unique identifier
- A set of outgoing edges
- A set of incoming edges
- A collection of properties (key-value pairs)

Each edge consists of:

- A unique identifier
- The vertex at which the edge starts (the *head vertex*)
- The vertex at which the edge ends (the *tail vertex*)
- A label to describe the kind of relationship between the two vertices
- A collection of properties (key-value pairs)

A graph can be stored as two tables, one for vertices and one for edges.

```
CREATE TABLE vertices (
  vertex_id integer PRIMARYKEY, properties json
);
CREATE TABLE edges (
  edge_id integer PRIMARY KEY,
  tail_vertex integer REFERENCES vertices (vertex_id),
  head_vertex integer REFERENCES vertices (vertex_id),
  label text,
  properties json
);

CREATE INDEX edges_tails ON edges (tail_vertex);
CREATE INDEX edges_heads ON edges (head_vertex);
```

Querying Graph Databases

- Property graph models, e.g. Cypher for Neo4J
- Declarative graph models, e.g. SPARQL
- Imperative graph query, e.g. Gremlin

Querying Graph Databases

Property Graph Model:

Both Nodes and Relationships can (should) have “names” (formally called labels for nodes and types for relationships), just like concepts and their relationships presented in the graph example. Relationships are directed, which is visualized by the arrowheads.

Both Nodes and Relationships may be associated with Properties, which are “key / value” pairs such as e.g. Color: Red. On the data model level, we call the key "Property Name”.

Source: https://neo4j.com/developer/graph-database/

Querying Graph Databases

- **Property graph models, e.g. Cypher for Neo4J**
- Declarative graph models, e.g. SPARQL
- Imperative graph query, e.g. Gremlin

Cypher Query Language

Neo4J uses Cypher as a declarative/property query language to search the content of graphs.

```
CREATE
  (NAmerica:Location {name:'North America', type:'continent'}),
  (USA:Location      {name:'United States', type:'country'  }),
  (Idaho:Location    {name:'Idaho',         type:'state'    }),
  (Lucy:Person       {name:'Lucy' }),
  (Idaho) -[:WITHIN]->  (USA)  -[:WITHIN]-> (NAmerica),
  (Lucy)  -[:BORN_IN]-> (Idaho)
```

find people who born in US and lives in Europe:

```
MATCH
  (person) -[:BORN_IN]->  () -[:WITHIN*0..]-> (us:Location {name:'United States'}),
  (person) -[:LIVES_IN]-> () -[:WITHIN*0..]-> (eu:Location {name:'Europe'})
RETURN person.name
```

Cypher Query Language

Neo4J uses Cypher as a declarative/property query language to search the content of graphs.

```
CREATE
  (NAmerica:Location {name:'North America', type:'continent'}),
  (USA:Location      {name:'United States', type:'country'  }),
  (Idaho:Location    {name:'Idaho',         type:'state'    }),
  (Lucy:Person       {name:'Lucy' }),
  (Idaho) -[:WITHIN]->  (USA)  -[:WITHIN]-> (NAmerica),
  (Lucy)  -[:BORN_IN]-> (Idaho)
```

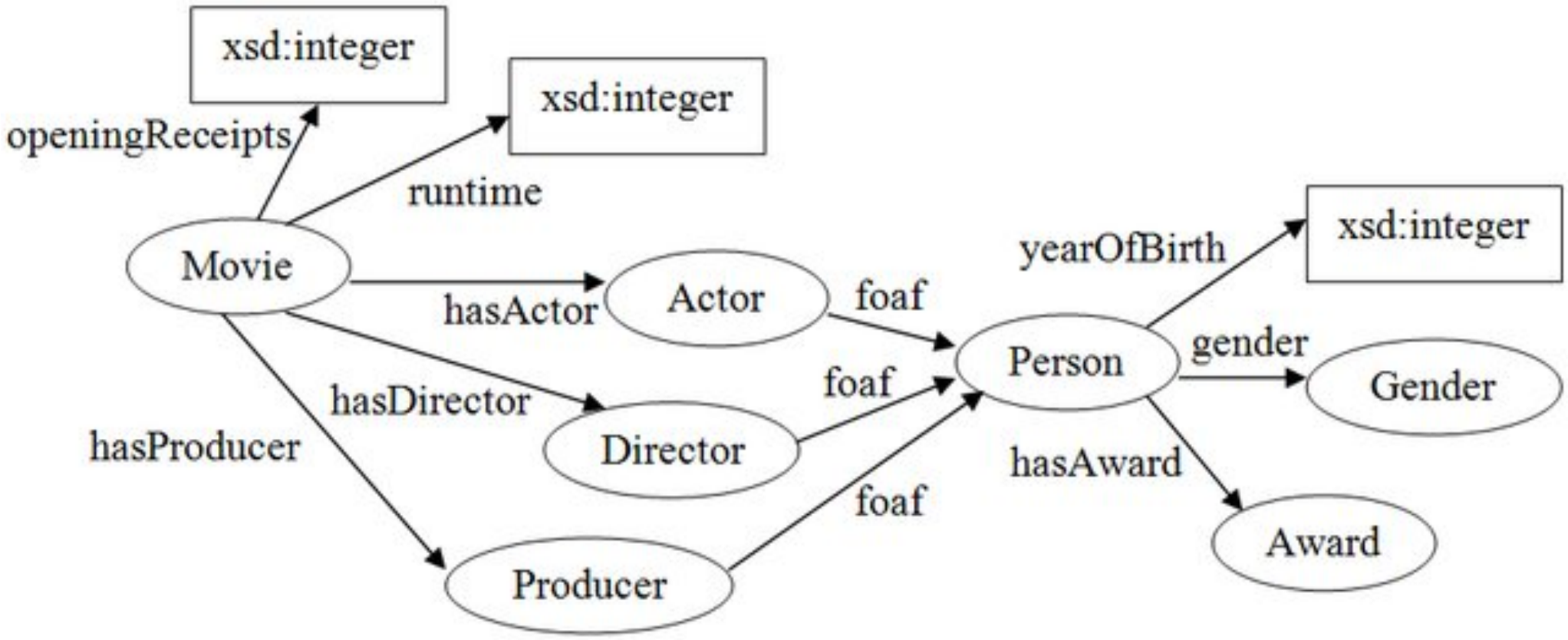
A chain of outgoing edges until the algorithm reaches the vertex of location

find people who born in US and lives in Europe:

```
MATCH
  (person) -[:BORN_IN]->  () -[:WITHIN*0..]-> (us:Location {name:'United States'}),
  (person) -[:LIVES_IN]-> () -[:WITHIN*0..]-> (eu:Location {name:'Europe'})
RETURN person.name
```

We can also start from the location and re-write the query.

RDF Example



subject, object, predicate

Lin, H. T., Koul, N., & Honavar, V. (2011). Learning Relational Bayesian Classifiers from RDF data. In *International Semantic Web Conference* (pp. 389-404). Springer, Berlin, Heidelberg.

SPAQL example

```
PREFIX : <urn:example:>

SELECT ?personName WHERE {
  ?person :name ?personName.
  ?person :bornIn  / :within* / :name "United States".
  ?person :livesIn / :within* / :name "Europe".
}
```

It operates on RDF data. RDF is a triple store graph which stores subject (vertex), object (vertex) and predicate (edge).
Another definition is that it stores **resource**, **property** and **property value**.

Cypher:

```
(person) -[:BORN_IN]-> () -[:WITHIN*0..]-> (location)
```

SPARQL:

```
?person :bornIn / :within* ?location.
```

Querying Graph Databases

- Property graph models, e.g. Cypher for Neo4J
- Declarative graph models, e.g. SPARQL
- **Imperative graph query, e.g. Gremlin**

Gremlin

Get the managers from Gremlin to the CEO in the hierarchy.

1. Get the vertex with the name "gremlin."
2. Traverse up the management chain
3. until a person with the title of CEO is reached.
4. Get name of the managers in the path traversed

```
g.V().has("name","gremlin").
  repeat(in("manages")).
    until(has("title","ceo")).
  path().by("name")
```

Get the 10 most central people in the knows-graph.

1. Get all people vertices.
2. Calculate their PageRank using knows-edges.
3. Order the people by their friendRank score.
4. Get the top 10 ranked people.

```
g.V().has("name","gremlin").
  out("knows").
  out("knows").
  values("name")
```

Outline

- File
- Databases
- Introduction to DBs
- Query Languages
- **SQL Database**
- NoSQL Database

MySQL

- Download from https://dev.mysql.com/downloads/mysql
- /usr/local/mysql… <— After installation in mac system located here.
- Give all permissions:
- `sudo chmod +rwx /usr/local/mysql/support-files/mysql.server`
- `sudo chmod +rwx /usr/local/mysql/bin`

1. `sudo /usr/local/mysql/support-files/mysql.server start`
2. **`mysql -u root -p`**
3. then enter this as pass: `root1234`

```
Add it to your PATH variable (.bash_profile):
export PATH="/usr/local/mysql/bin:$PATH"
```

My password: root1234

Download the JDBC Driveer

Download and add the following file to your JAR files:

mysql-connector-j-8.0.31.jar

MySQL

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
```

```
mysql> create database met622;
Query OK, 1 row affected (0.00 sec)


mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| met622             |
| mysql              |
| performance_schema |
| sys                |
+--------------------+

mysql> use met622;
Database changed
```

SQL Example

```
mysql> SHOW TABLES;
Empty set (0.00 sec)


mysql> CREATE TABLE Pet (name VARCHAR(20), owner VARCHAR(20), species
VARCHAR(20), sex CHAR(1));
Query OK, 0 rows affected (0.01 sec)

mysql> Show tables;
+------------------+
| Tables_in_met622 |
+------------------+
| pet              |
+------------------+
1 row in set (0.00 sec)

mysql> Describe Pet;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| name    | varchar(20) | YES  |     | NULL    |       |
| owner   | varchar(20) | YES  |     | NULL    |       |
| species | varchar(20) | YES  |     | NULL    |       |
| sex     | char(1)     | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
4 rows in set (0.01 sec)
```

SQL Example

```
mysql>  insert into Pet values("puffball","nobody","hamster","M");
Query OK, 1 row affected (0.00 sec)

mysql>  insert into Pet values(“misfat”,”me”,”cat”,”F”);
Query OK, 1 row affected (0.00 sec)

mysql> select * from Pet;
+-----------+--------+----------+------+
| name      | owner  | species | sex  |
+-----------+--------+----------+------+
| puffball | nobody | hamster | M    |
| misfat   | me     | cat     | F    |
+----------+--------+----------+------+
2 rows in set (0.00 sec)
```

Some SQL commands

- SELECT * FROM Pet;
- SELECT ‘name’, ‘gender' FROM Pet;
- INSERT INTO Pet VALUES('sunny','neighbor','dog','M');
- ALTER TABLE table_name ADD column_name datatype;
- ALTER TABLE table_name DROP column_name;
- CREATE TABLE EMPLOYEE (Name VARCHAR2(20), Email VARCHAR2(100), DOB DATE);
- DROP TABLE EMPLOYEE;
- INSERT INTO TABLE_NAME (col1, col2, col3,.... col N) VALUES (value1, value2, value3, ... valueN);
- UPDATE table_name SET [column_name1= value1,...column_nameN = valueN] [WHERE CONDITION]
- UPDATE Pet SET User_Name = 'foo' WHERE owner = 'me'

SQL Query Class Implementation

- Find the pets, which their gender are F.
- Add into the pet table a new pet, with a data you like.
- Update the name of the pet you have added into the table with <oldname>2.

JDBC

- Add the **mysql-connector-java-8.0.16.jar** into your lib folder, and also define for your IDE where is your lib folder.

- Be sure that the database is running and connected, e.g. mysql server should be running.

- Establishes a connection to the database.

- Create the SQL command to be executed on the database.

- Execute the statement and if it is a query stores its result in a result set.

- Do what ever you need with the result set.

- Close the connection <— don’t forget it.

SQL Select with JDBC

```
package edu.bu.met622.db.mysql;
import java.sql.*;


public class JDBCTest {
 public static void main(String[] args) {
   try {
      Connection con = DriverManager.getConnection("jdbc:mysql://
localhost:3306/met622?useTimezone=true&serverTimezone=UTC","root", “root1234");
      System.out.println("--->"+con.toString());
      Statement stmt=con.createStatement();
      ResultSet rs=stmt.executeQuery("select * from pet");
      while(rs.next())  {
         System.out.println(
              “row:”+rs.getString(1)+","+ rs.getString(2)+","+rs.getString(3));
      }
      con.close();
   }catch (Exception ex){
      ex.printStackTrace();
   }
 }
}
```

SQL Insert with JDBC

```
 private void insert(String name, String owner,String race, Character gender)
{ try {
        Connection con = DriverManager.getConnection(“…”,”root”,"root");
        System.out.println("--->"+con.toString());

        Statement stmt = con.createStatement();
         System.out.println("----->"+ "INSERT INTO pet(name,owner,species,sex) VALUES
(‘"+name+"','"+owner+"','"+race+"','"+gender+"')");

     stmt.executeUpdate("INSERT INTO pet(name,owner,species,sex)  VALUES
('"+name+"','"+owner+"','"+race+"','"+gender+"')");

      con.close();
     }catch (Exception ex) {
       ex.printStackTrace();
     }
 }
```

SQL Delete with JDBC

```
private void delete(String name, String owner) {
   try {
      Connection con = DriverManager.getConnection(“jdbc:mysql://
          localhost:3306/met662","root","root");
      System.out.println("--->"+con.toString());
      Statement stmt = con.createStatement();
       System.out.println("DELETE FROM pet WHERE name='"+name+"' AND
owner='"+owner+"'");
      stmt.executeUpdate("DELETE FROM pet WHERE name='"+name+"' AND
owner='"+owner+"'");
      con.close();
    } catch (Exception ex) {
       ex.printStackTrace();
    }
 }
```

JDBC implementation example

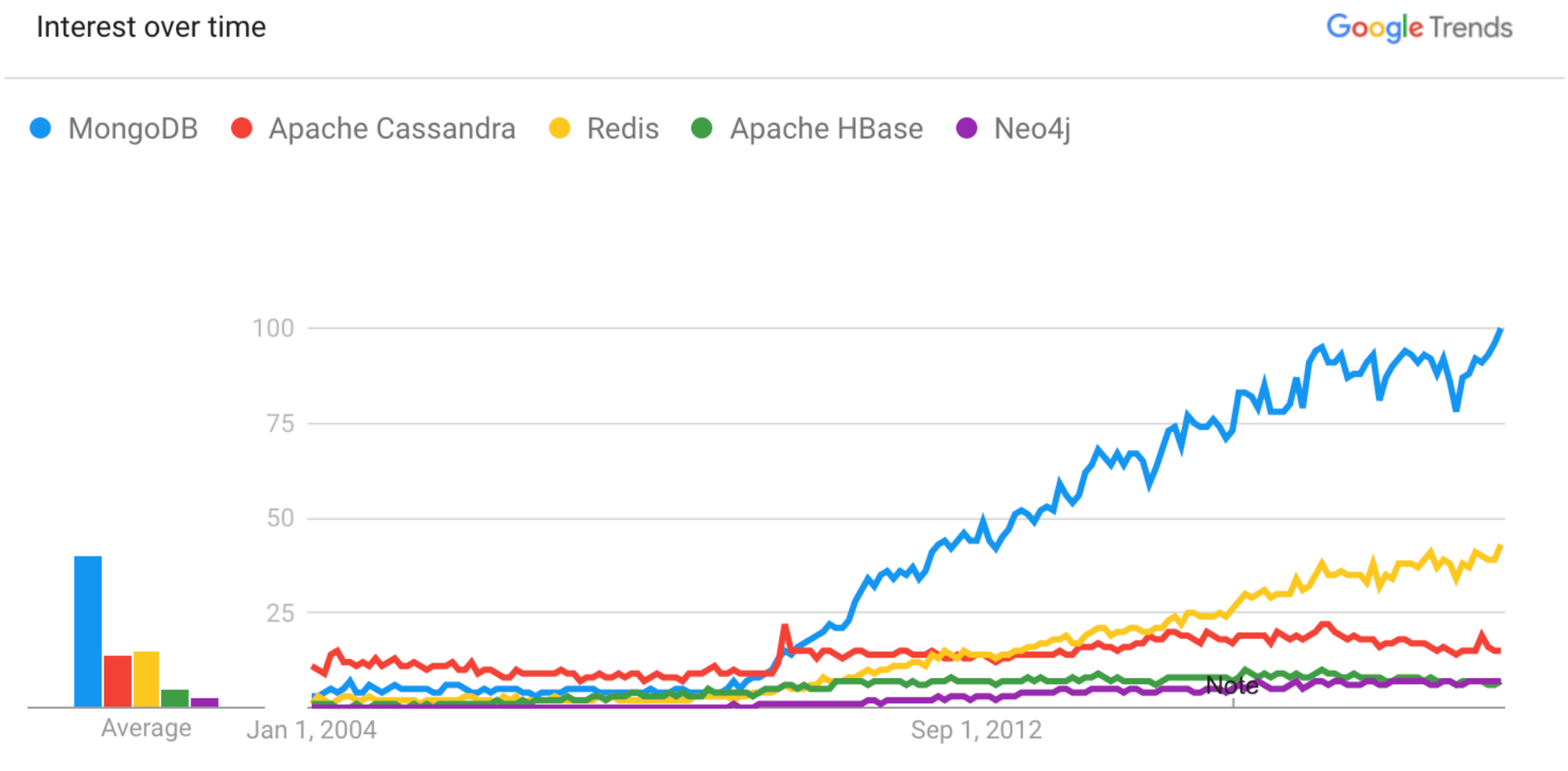
- Find the pets, which their gender are F.
- Add into the pet table a new pet, with a data objects you like.
- Update the name of the pet you have added into the new table with <oldname>2.

Outline

- File
- Databases
- Introduction to DBs
- Query Languages
- SQL Database
- **NoSQL Database**

MongoDB

- MongoDB is the most popular NoSQL, or document databases.



MongoDB

- MongoDB is an open source database system.
- MongoDB stores the data in JSON like document. It means the document structure can change and there is no specific schema that all documents should follow.
- Unlike relational model, document models are easier to be used inside application.
- A record in MongoDB is a document, which includes filed and value pairs, similar to JSON object.
- It introduces another format, BSON (Binary JSON), which can hold larger documents as well.

MongoDB Terms

- Instead of **Table** in MySQL we have **Collection** in MongoDB.
- Instead of **Row** in MySQL we have **Document** in MongoDB.
- Instead of **Column** in MySQL we have **Filed** in MongoDB.

Installing MongoDB

- Use this link: https://www.mongodb.com/docs/manual/tutorial/install-mongodb-on-os-x/
- Some extra command that help me installing MongoDB on M1

```
sudo chown -R $(whoami):admin /usr/local/* && sudo chmod -R g+rwx /usr/local/*

/usr/sbin/softwareupdate --install-rosetta –agree-to-license

arch -x86_64 /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install.sh)”

arch -x86_64 brew install mongodb-community@6.0
```

- Add this jar files into your classpath:
  - `mongo-java-driver-3.12.11.jar`

Running MongoDB

1. `mongod --config /usr/local/etc/mongod.conf`
2. `brew services start mongodb-community@6.0`
3. Mongosh

- If you can't run `mongo` from console go inside the following folder and then try to run `mongo` command from there:

`/usr/local/Cellar/mongodb-community/6.0.1/bin`

- For windows OS try to locate it in one of your "`Program Files`" folder

MongoDB Data Types

String – The most commonly used datatype to store the data.
Integer –
Boolean –
Double –
Min/ Max keys – It is used to compare a value against the lowest and highest BSON elements.
Arrays – It is used to store arrays or list or multiple values into one key.
Timestamp –
Object – This datatype is used for embedded documents.
Null –
Symbol – Reserved for languages that use a specific symbol type.
Date – Stores the current date or time in UNIX time format.
Object ID – Stores the document's ID.
Binary data – It is used to store binary data, e.g. music, picture and movies.
Code – It is used to store JavaScript code inside a MongoDB document.
Regular expression – it is used to store regular expression.

MongoDB Commands

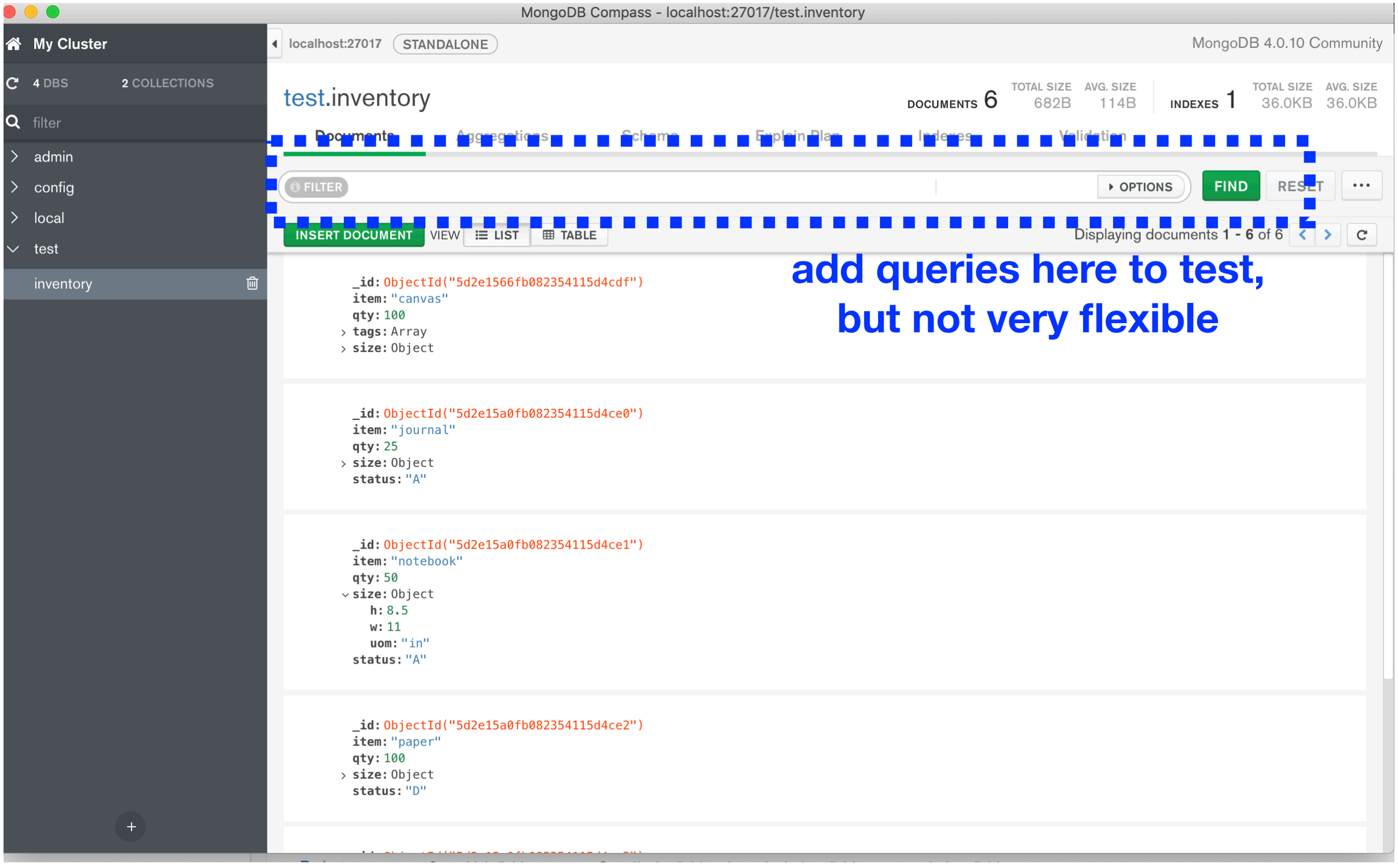
```
show databases, show dbs

db
```

`use test` <= this command creates a database, but it does not shown in the list, until this database has a collection added to it.

```
db.help()

db.stats()
```


MongoDB Compass - localhost:27017/test.inventory
My Cluster
4 DBS
2 COLLECTIONS
filter
admin
config
local
test
inventory
localhost:27017
STANDALONE
MongoDB 4.0.10 Community
test.inventory
DOCUMENTS 6
TOTAL SIZE 682B
AVG. SIZE 114B
INDEXES 1
TOTAL SIZE 36.0KB
AVG. SIZE 36.0KB
Documents
Aggregations
Schema
Explain Plan
Indexes
Validation
FILTER
OPTIONS
FIND
RESET
INSERT DOCUMENT
VIEW
LIST
TABLE
Displaying documents 1 - 6 of 6
add queries here to test,
but not very flexible
_id: ObjectId("5d2e1566fb082354115d4cdf")
item: "canvas"
qty: 100
tags: Array
size: Object
_id: ObjectId("5d2e15a0fb082354115d4ce0")
item: "journal"
qty: 25
size: Object
status: "A"
_id: ObjectId("5d2e15a0fb082354115d4ce1")
item: "notebook"
qty: 50
size: Object
h: 8.5
w: 11
uom: "in"
status: "A"
_id: ObjectId("5d2e15a0fb082354115d4ce2")
item: "paper"
qty: 100
size: Object
status: "D"

MongoDB Commands

- `db.createCollection(“mycollection")`
- `db.mycollection.insert({test:”1”})`
- **`show collections`**
- **`db.inventory.find({})`**
- `db.mycollection.drop()`

Inserting a Document into a Collection

```
db.COLLECTION_NAME.insert(DOCUMENT)
```

- `db.inventory.insert({test:'mock data'})`

Inserting a new Document

```
db.inventory.insertOne(
   { item: "canvas",
     qty: 100,
     tags: ["cotton"],
     size: { h: 28, w: 35.5, uom: "cm" } }
)
```

```
{
   "acknowledged" : true,
   "insertedId" : ObjectId("5d2e1566fb082354115d4cdf")
}
```

Insert more than one Document

```
db.inventory.insertMany([
   { item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status: "A" },
   { item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" }, status: "A" },
   { item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status: "D" },
   { item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" }, status: "D" },
   { item: "postcard", qty: 45, size: { h: 10, w: 15.25, uom: "cm" }, status: "A" }
]);
```

```
"acknowledged" : true,
"insertedIds" : [
        ObjectId("5d2e15a0fb082354115d4ce0"),
        ObjectId("5d2e15a0fb082354115d4ce1"),
        ObjectId("5d2e15a0fb082354115d4ce2"),
        ObjectId("5d2e15a0fb082354115d4ce3"),
        ObjectId("5d2e15a0fb082354115d4ce4")
]
```

Remove Document from Collection

```
db.COLLECTION_NAME.remove(DELETE_CRITERIA)
```

- `db.inventory.remove({test:'mock data'})`
- `db.inventory.find({})`
- `db.inventory.remove({})` <— removes all documents in a collection.

MongoDB Query

- In SQL we wrote “**SELECT** * **FROM** inventory **WHERE** {SEARCH_CRITERIA} ”
- in MongoDB we should write:

```
db.COLLECTION_NAME.find({SEARCH_CRITERIA})
```

- If there is a criteria we write it inside the bracket for find.
  `db.inventory.find( { status: "D" } )`
- `db.inventory.find( {"qty":100 } )`
- `db.inventory.find( { status: { $in: ["A","D"]}})`
- `db.inventory.find( { status: "A", qty: {$lt:30}})`

More Complex queries

- SELECT * FROM inventory WHERE status = "A" OR qty < 30

```
db.inventory.find( { status: "A",$or: [ { qty: { $lt: 30 } }, { item: /
^p/ } ] } )
```

- SELECT * FROM inventory WHERE status = "A" AND (qty < 30 OR item LIKE “p%")

```
db.inventory.find( { status: "A", $and: [ { qty: { $lt: 30 } },
{ item: /^p/ } ] } )
```

MongoDB Query

- In SQL we wrote “**SELECT** * **FROM** inventory”

- in MongoDB we should write:
  `db.inventory.find( {} )`

- If there is a criteria we write it inside the bracket for find.
  `db.inventory.find( { status: "D" } )`

-
-

```
"item": "journal", "qty": 25, "size": { "h":14, "w":21, "uom":"cm" }, "status":"A" }
```

- `db.inventory.find( { status: "A", qty: {$lt:30}})`

MongoDB JAR File

- mongodb-driver-3.4.3.jar
- mongodb-driver-core3.4.3.jar
- bson-3.4.3.jar

Using MongoDB inside JAVA

```java
// initialize the client object
MongoClient mongoClient = new MongoClient();
// get the 'test' dataset
MongoDatabase dbObj = mongoClient.getDatabase( "test" );

 // list its collections
for (String name : dbObj.listCollectionNames()) {
 System.out.println("Collections inside this db:"+name);
}

// Or explicitly we can go to its collection
MongoCollection<Document> col = dbObj.getCollection("inventory");

 // how to read the content of a document.
Iterator it = col.find().iterator();
while (it.hasNext()) {
 System.out.println("docs inside the col:" +it.next());
}
```

Using MongoDB inside JAVA

```
dbObj.createCollection("newcolzero");

MongoCollection<Document> col =
dbObj.getCollection("newcolzero");
        // insert new document into a
collection
        Document mydoc = new Document();
        mydoc.put("name", "johnzzzz");
        mydoc.put("role", "studentzzz");
        col.insertOne( mydoc );
```

Using MongoDB inside JAVA

```
//remove the collection
MongoCollection<Document> collection=
                                    dbObj.getCollection("newcolzero");
collection.drop();

//remove a document from a collection
BasicDBObject doc2remove = new BasicDBObject();
doc2remove.put("name","john");
col.deleteMany(doc2remove);
```

Let’s Implement more MongoDB example in the class (Recommend me an example)

References

- https://docs.mongodb.com/manual/tutorial/getting-started/
- https://docs.mongodb.com/v3.2/tutorial
- https://www.w3schools.com/sql/