ULB

UNIVERSITÉ LIBRE DE BRUXELLES, UNIVERSITÉ D'EUROPE

INFO-H-400
Medical Information Systems

Prof. David Wikler (dwikler@ulb.ac.be)

Teaching Assistant: Ir Adrien Foucart (afoucart@ulb.ac.be)

Course organisation

- 5 ECTS : 2 course, 2 practical work, 1 project
- Evaluation :
  - Oral Exam : 3 ECTS (2 course + 1 practical work)
  - Lab reports : 1 ECTS
  - Project : 1 ECTS
- Slides and handouts on the Virtual University (http://uv.ulb.ac.be)

Lab topics

- Java and Databases (3 labs)
  - Building a basic (medical) software with a relational database.
- HL7 (1 lab)
  - Using HL7 to communicate patient/medical information.
- DICOM (3 labs)
  - Using the DICOM standard to store and transfer radiology information and images.
- FHIR (2 labs)
  - Creating RESTful client/server applications for medical information.
- eID (1 lab)
  - Using the electronic ID card for authentication in a software.

Lab Sessions

- **Before the lab:** preparation documents available online on the Virtual University.
  - Contains informations on the content of the lab, and questions to guide the work. **The preparation must be done before the lab.**
- **During the lab:** problem to solve
  - Developping a JAVA application using the different technologies seen during the course (DICOM, HL7, eID, etc…)
- **After the lab:** report to upload on the Virtual University

Lab reports

- The report must **explain the theory using the practical work**.
- The **preparation questions** are useful as a guide, but the report shouldn't just be a series of questions & answers.
- The report must be a short **synthesis** (1-2 pages) of the different concepts illustrated in the lab.
- The report can be written in French or in English.

Project

- Project in small groups (5-6 persons)
- General requirements of a Health Information System to design & implement.
- Project goals:
  - To define the **requirements** and the **feature list**.
  - To **divide the tasks** within the group and **implement the software**.
  - Pay attention to the interactions between the different parts of the project!
- 2x4h + 3x2h to work on the project in the LISA computer room.
- Oral defense at the end of the project.

ULB

UNIVERSITÉ LIBRE DE BRUXELLES, UNIVERSITÉ D'EUROPE

Building a (medical) software

Introduction to software and database design for medical information systems

Building a (medical) software

- Software design
- Databases design
  - Organizing the data
  - Building the database
  - SQL introduction
- Java – what is it again?

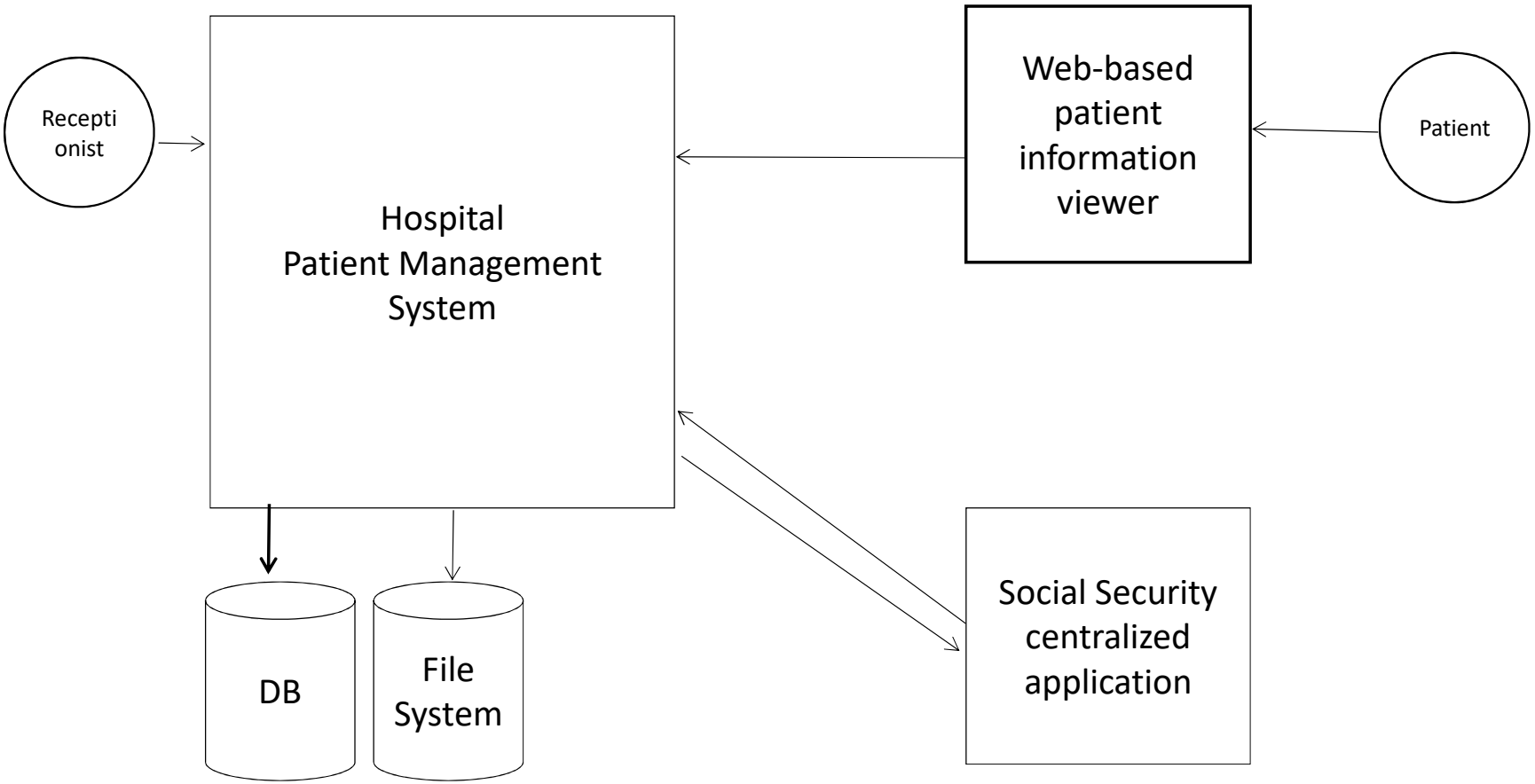
Building a software

- Listen to the **demands** and **needs** of the client.
- Define the **requirements** that would fulfill those needs.
- Define the **feature list** for those requirements.
- Design the **data** model and the **software** model.
- **Implement** the features.

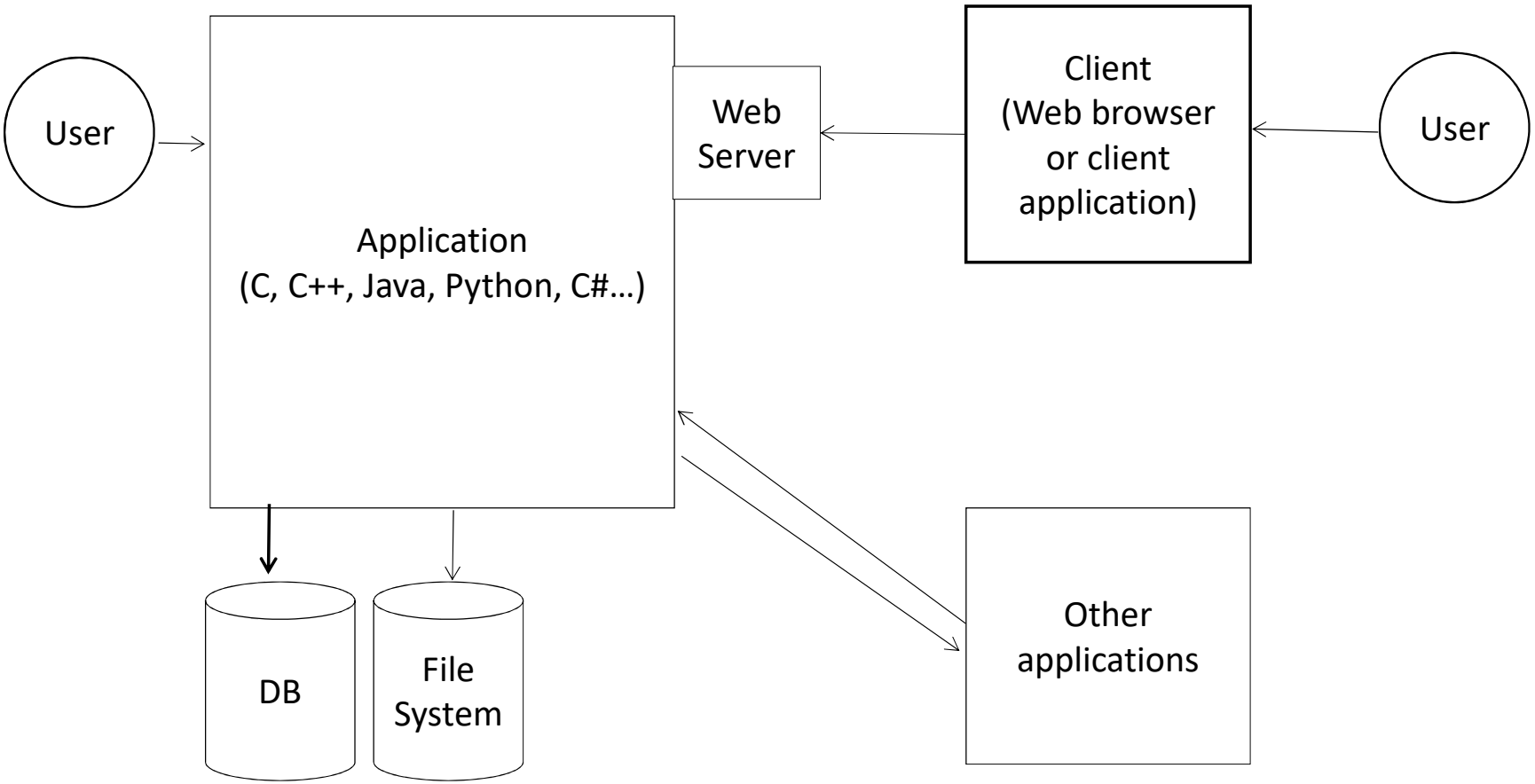
A note on personal/medical data storage

- The storage of sensitive personal (and medical) data is regulated at the European and Belgian level.
  - "*Loi relative à la protection de la vie privée à l'égard des traitements de données à caractère personnel*" (1992)
  - Belgian laws relative to patient rights and to the Electronic Health Record.
  - eHealth guidelines
  - "*General Data Protection Regulation (GDPR)*" (European regulation, new rules in May 2018)
- Main idea:
  - The patient must be able to access *all data related to him/her*.
  - The patient can *grant/revoke* access to his/her data.
  - The patient must *explicitly consent* to any data gathering and usage.
  - All personal data must be safely stored/transmitted (encrypted).
  - All personal data access must be *justified* and *logged*.

A very generic architecture



A very generic architecture



Software vocabulary

- A **server** is an application that provides a service or a ressource over, usually over the Internet, to a **client** application using an agreed-upon **communication protocol**.
  - E.g. a **web server** answers requests from a **browser** using **HTTP(S)**.
- A **database** is a collection of data.
- A **database management system** (DBMS) is a software which organizes the storage and operation of the database.

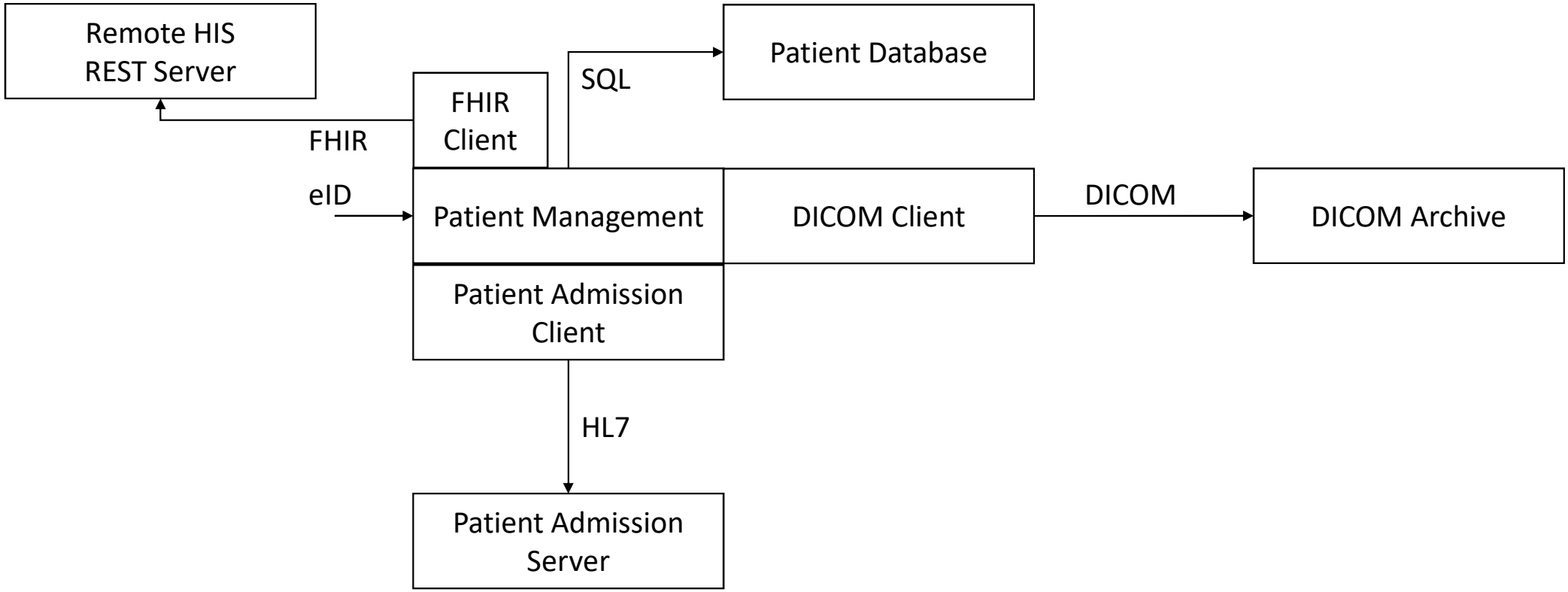
Building a software – the labs

- "LISA Health Information System" requirements
  - A system to **store and manage patient data**.
  - Which can handle **patient admission** from multiple computers.
  - Which can be linked to an **image archive** to get & view images.
  - Which can communicate patient data with another HIS.

Building a software – the labs

- "LISA Health Information System" requirements
  - A system to **store and manage patient data**.
  - Which can handle **patient admission** from multiple computers.
  - Which can be linked to an **image archive** to get & view images.
  - Which can communicate patient data with another HIS.
  - (Which illustrates the different concepts & technologies seen during the courses)

Building a software – the labs



Building a (medical) software

- Software design
- Databases design
  - Organizing the data
  - Building the database
  - SQL introduction
- Java – what is it again?

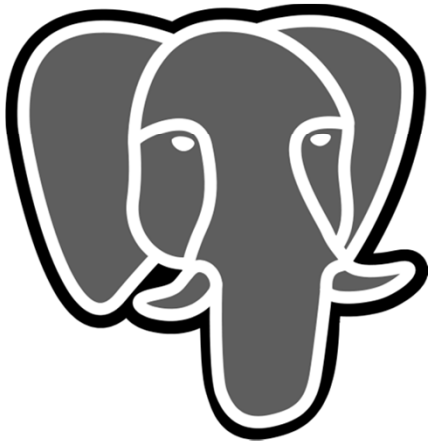
Medical data

- Personal information (name, address, date of birth…)
  - Text, dates…
- Medical history (past diagnostics, treatments…)
  - Text, structured/semantic information
- Imaging (radiology, MRI…)
  - Images, metadatas
- Drugs (prescriptions)
  - Structured/semantic information

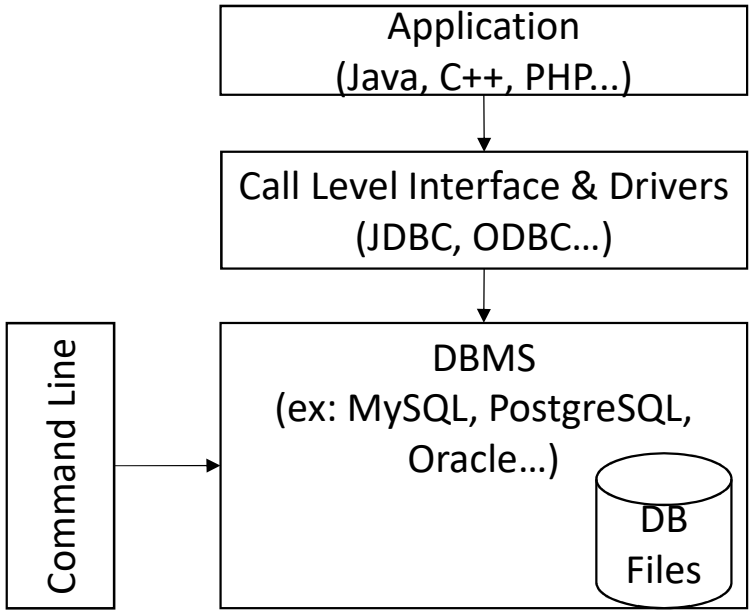
DataBase Management System

- Goals:
  - **Organize** the data in a coherent manner.
  - **Keep** the data for as long as necessary, **securely**. Prevent **data loss**.
  - Control **retrieval**, **insertion** and m**odification** of the data, on demand, **securely**.
- The **DBMS** is a software providing tools for the creation of the database, the management of access rights, and the management of all possible operations on the data.

DBMS



DBMS



- One database can be accessed by different means simultaneously.
- "Terminal" access (command line).
- "Graphic" access (admin software like PHPMyAdmin, MySQL Workbench, PgAdmin…)
- "Programming" access from a software using the CLI to read/write the data.

DBMS

- The "**Call Level Interface**" (CLI) provides a way for other softwares to manipulate the database.
- **Java Database Connectivity** (JDBC) is the programming interface designed for Java applications to operate different database management systems using a common interface.

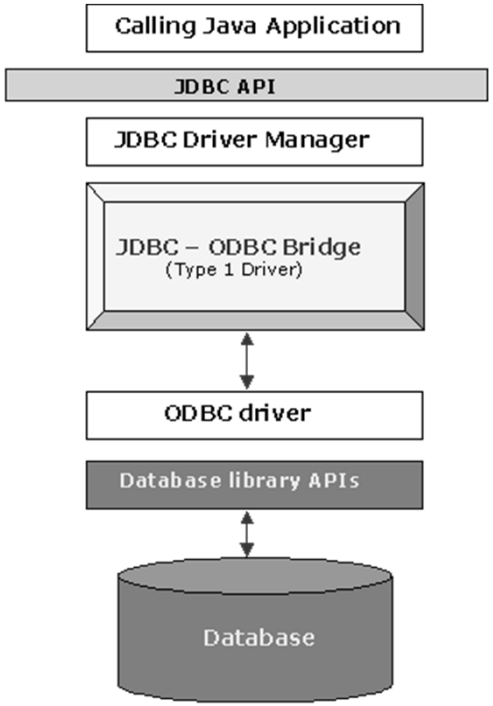


Image: Wikimedia Commons

Building a (medical) software

- Software design
- Databases design
  - Organizing the data
  - Building the database
  - SQL introduction
- Java – what is it again?

Organizing the data

Patient appointments

Monday, April 23rd, 2018	Tuesday, April 24rd, 2018	Wednesday, April 25th, 2018
8h10 Foucart(Dr A) 8h25 Wikler(Dr B) 9h30 Schenkel(Dr B) 11h15 Debeir(Dr A) 14h30 Decaestecker(Dr A)	9h10 Teller (Dr C) 12h30 Baele(Dr B) 12h30 Hahn (Dr C)	14h00 Warzée(Dr A) 15h00 Grave (Dr C) 15h00 VanEyke(Dr A) 16h15 Lafruit(Dr C)

- What kind of data do we have?
  - **Dates** et **hours** of **appointments**.
  - **Names** of **patients**.
  - **Names of doctors**.

Organizing the data

Patient	Doctor	Appointment
Foucart	Dr A	Monday April 23rd, 2018, 8h10
Wikler	Dr B	Monday April 23rd, 2018, 8h15
...	...	...
Van Eyke	Dr A	Wednesday April 25th, 2018, 15h00
Lafruit	Dr C	Wednesday April 25th, 2018, 16h15

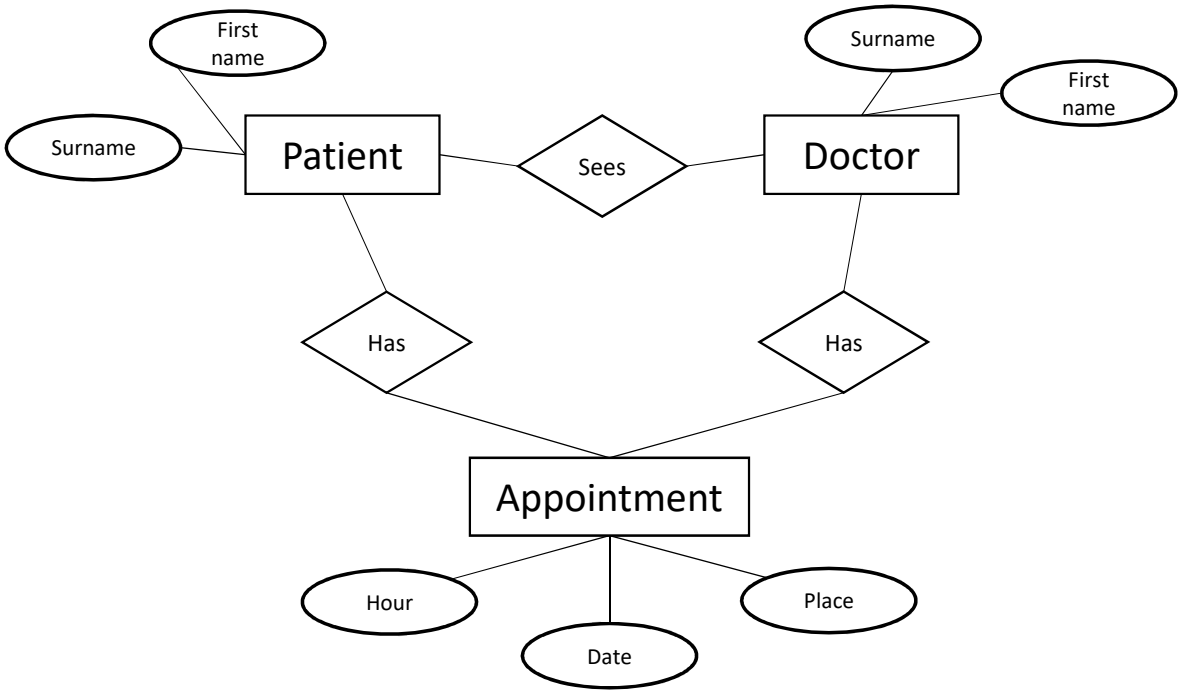
- What happens if we want to add information on a patient (first name, address, phone, national registry number…) or a doctor (INAMI number…)?
- What happens if we want to update the information?

Organizing the data

- Identify the **entities** and their **attributes**.
  - **Dates** and **hours** of **appointments**.
  - **Names** of **patients**.
  - **Names** of **doctors**.
- Identify the **relationships** between **entities**.
  - The **patient**, who has a **surname** and a **first name**, **sees** the **doctor**, who has a **surname** and a **first name**, during an **appointment**, which has an **hour**, a **date** and a **place**.

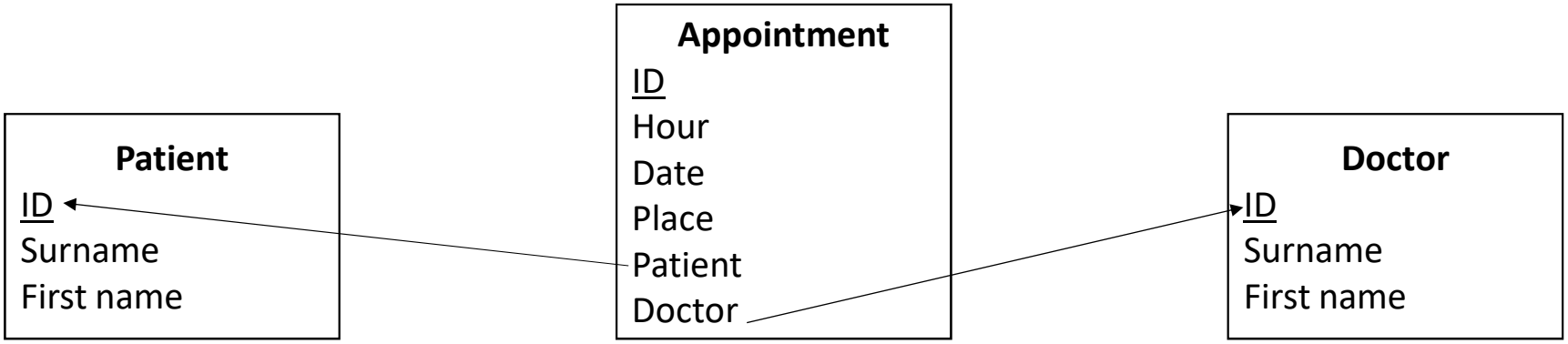
Organizing the data

- We can use an **entity-relationship diagram** to represent the relationship between those entitites.



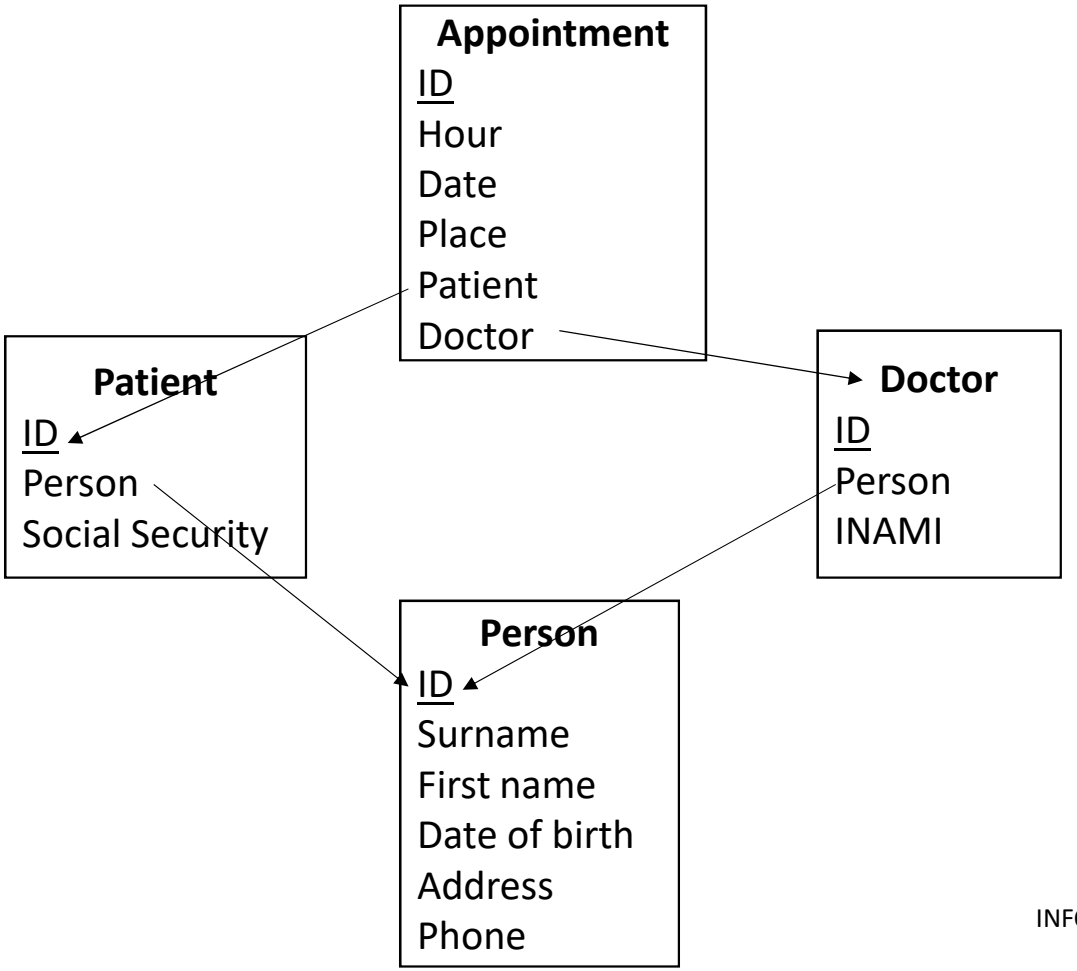
Organizing the data

- From the **entity-relationship** diagram, we can create the **relational diagram** of the data.



- What happens if a doctor is also patient of another doctor?

Organizing the data



An **appointment gathers** a **patient** and a **doctor** in a certain **place**, at a certain **date** and **hour**.

A **doctor is** a **person** with an **INAMI number**.

A **patient is** a **person** with a **social security number**.

A **person** has a **surname** and a **first name**, a **date of birth**, an **address**, and a **phone number**.

Organizing the data

ID_PER	Surname	First name	…	Phone
1	Foucart	Adrien	…	02/6502297
2	A	Docteur	…	01/2345678

ID_PAT	Social security	ID_PER
1	123456-789-555	1

ID_MED	INAMI	ID_PER
1	1 2345678 90	2

ID_APP	Hour	Date	Place	ID_PAT	ID_MED
1	8h10	3/4/2017	Cabinet A	1	1

Organizing the data

- The tables and the links between them form the **database schema**.
- The schema must be designed keeping in mind the **possible future modifications** (is it easy to add a property?) and **extension** (is it easy to reuse the tables for other applications? Is it easy to link new tables to existing ones?)

Building a (medical) software

- Software design
- Databases design
  - Organizing the data
  - Building the database
  - SQL introduction
- Java – what is it again?

Creating the database

- Each **table** must have a **unique name** and contains data about entities of the same nature.
- Each **row** of a table describes the data of **one entity**, each **column** describes **one attribute**.
- Rows in a table are **distinct** and **not empty**.
- The **content** of a database must be easy to modify frequently. The **schema** must be as fixed as possible.

Creating the database

- An attribute must a **as simple as possible**. Complex attributes must be split.

Name	**Contact**
Adrien Foucart	Av. F.D. Roosevelt 50, 1050 Ixelles Tél:026502297 Email: afoucart@ulb.ac.be

Surname	**First name**	**Street**	**ZIP code**	**Phone**	**Email**
Foucart	Adrien	Av FD Roosevelt	1050	026502297	afoucart@ulb.ac.be

Creating the database

- If a table has **functional dependency** between attributes, those attributes should (usually) form a new table.

Surname	First name	Service	University	Service web page
Foucart	Adrien	LISA	ULB	http://lisa.ulb.ac.be

Surname	First name	ID_Service
Foucart	Adrien	1

ID_Service	Service	University	Service web page
1	LISA	ULB	http://lisa.ulb.ac.be

Creating the database

- The **schema** of a database specifies:
  - The **name** of the tables
  - The **name**, **type** and **optionality** of each attribute
  - The **primary key** (column or group of column with a **unique**, **immutable**, **mandatory** value)
  - The optional **foreign keys**, associated with relationship constraints (usually, on the primary key of another table)

Creating the database

Appointment
ID (INTEGER PRIMARY KEY)
Hour (TIME)
Date (DATE)
Place (STRING)
Patient (INTEGER FOREIGN KEY -> PATIENT.ID)
Doctor (INTEGER FOREIGN KEY -> DOCTOR.ID)

Patient
ID (INTEGER PRIMARY KEY)
Person (INTEGER FOREIGN KEY -> PERSON.ID)
Social Security (STRING OPTIONAL)

Doctor
ID (INTEGER PRIMARY KEY)
Person (INTEGER FOREIGN KEY -> PERSON.ID)
INAMI (STRING)

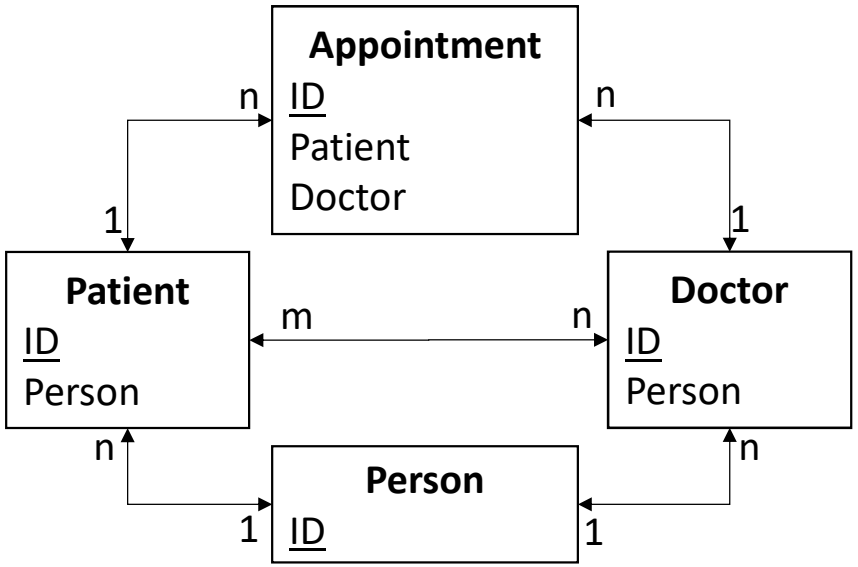
Person
ID (INTEGER PRIMARY KEY)
Surname (STRING)
First name (STRING)
Date of birth (DATE)
Address (STRING OPTIONAL)
Phone (STRING OPTIONAL)

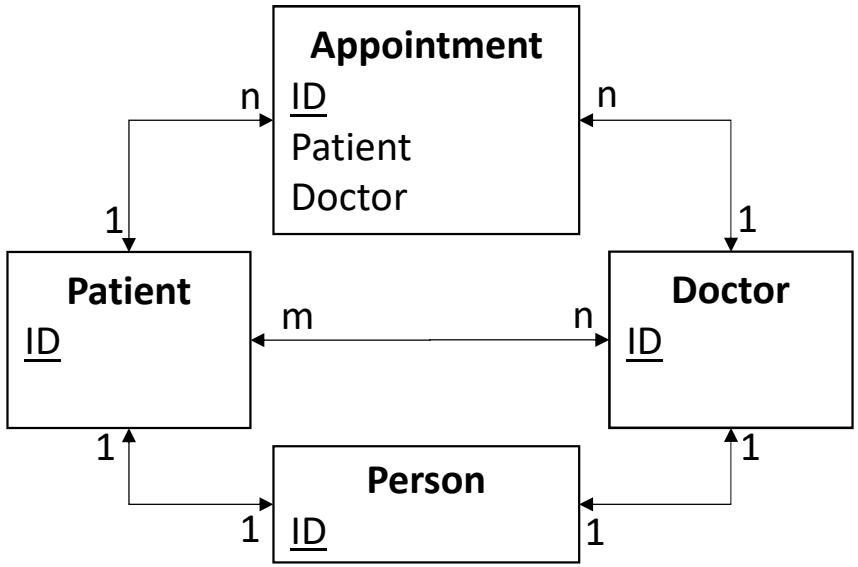
Creating the database

- Relationships between entities have a **cardinality**.
  - 1-n: one entity is linked to n entities.
    - E.g. **one patient** has **n appointments**
  - 1-1: one entity is linked to 1 entity.
    - E.g. **one patient** is **one person**
  - m-n: m entities are linked to n entities.
    - E.g. **m patients** consult **n doctors**.

Creating the database

- The cardinality determines how the relationship model must be built.





Patient.ID = Person.ID Patient.ID = Doctor.ID

Creating the database

- From the relational schema, we can write (or automatically generate) code that the DBMS will execute to create the database.

```
CREATE TABLE IF NOT EXISTS 'Person' (
'id_per' INTEGER AUTO_INCREMENT  UNIQUE,
'surname' VARCHAR(200) NOT NULL,
'first_name' VARCHAR(200) NOT NULL,
'dob' DATE NOT NULL,
PRIMARY KEY ('id_per') );
```

```
CREATE TABLE IF NOT EXISTS 'Patient' (
'id_pat' INTEGER AUTO_INCREMENT  UNIQUE,
'socialSecurity' VARCHAR(200) DEFAULT NULL,
'id_per' INTEGER NOT NULL UNIQUE,
PRIMARY KEY ('id_pat') );

ALTER TABLE 'Patient' ADD FOREIGN KEY('id_per') REFERENCES 'Person'.'id_per';
```

Building a (medical) software

- Software design
- Databases design
  - Organizing the data
  - Building the database
  - SQL introduction
- Java – what is it again?

SQL introduction

- **SQL** (Structured Query Language) is a normalized language to execute operations on databases.
- The language has four parts:
  - Data definition (CREATE TABLE, CREATE DATABASE…)
  - Data manipulation (INSERT, UPDATE, DELETE…)
  - Transaction control (START TRANSACTION, COMMIT, ROLLBACK…)
  - Data control (GRANT, DENY…)
- It is used by the vast majority of existing DBMS (with minor variations).
- It is a declarative language.

SQL introduction

```
E:\xampp\mysql\bin>mysql -u student -p
```

CREATE DATABASE test;
SHOW DATABASES;
DROP DATABASE test;
USE test;

```
MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| medical            |
| test               |
+--------------------+
```

```
CREATE TABLE person (
        id_per INTEGER NOT NULL AUTO_INCREMENT,
        surname VARCHAR(200) NOT NULL,
        first_name VARCHAR(200) NOT NULL,
        dob DATE NOT NULL,
        PRIMARY KEY(id_per) );
```

```
MariaDB [test]> describe person;
+------------+--------------+------+-----+---------+----------------+
| Field      | Type         | Null | Key | Default | Extra          |
+------------+--------------+------+-----+---------+----------------+
| id_per     | int(11)      | NO   | PRI | NULL    | auto_increment |
| surname    | varchar(200) | NO   |     | NULL    |                |
| first_name | varchar(200) | NO   |     | NULL    |                |
| dob        | date         | NO   |     | NULL    |                |
+------------+--------------+------+-----+---------+----------------+
4 rows in set (0.04 sec)
```

SQL introduction

```
INSERT INTO person(surname,first_name,dob)
VALUES
("Foucart", "Adrien", "1988-04-11"),
("Dupont", "Jean", "1990-08-24");
```

```
MariaDB [test]> select * from person;
+--------+---------+------------+------------+
| id_per | surname | first_name | dob        |
+--------+---------+------------+------------+
|      1 | Foucart | Adrien     | 1988-04-11 |
|      2 | Dupont  | Jean       | 1990-08-24 |
+--------+---------+------------+------------+
2 rows in set (0.00 sec)
```

```
SELECT * FROM person;
SELECT surname,dob FROM person;
```

```
MariaDB [test]> SELECT surname,dob FROM person;
+---------+------------+
| surname | dob        |
+---------+------------+
| Foucart | 1988-04-11 |
| Dupont  | 1970-08-24 |
+---------+------------+
2 rows in set (0.00 sec)
```

```
UPDATE person
SET dob = "1970-08-24"
WHERE id_per = 2;
```

SQL introduction

DELETE FROM person
WHERE id_per = 1;

```
MariaDB [test]> DELETE FROM person WHERE surname LIKE "Fou%";
Query OK, 1 row affected (0.06 sec)

MariaDB [test]> SELECT * FROM person;
+--------+---------+------------+------------+
| id_per | surname | first_name | dob        |
+--------+---------+------------+------------+
|      2 | Dupont  | Jean       | 1970-08-24 |
+--------+---------+------------+------------+
1 row in set (0.00 sec)
```

...WHERE `**price**` <= 99,9 ;
...WHERE `**age**` BETWEEN 12 AND 16 ;
...WHERE `**color**` IN (`blue`, `red`);
...WHERE `**serie**` LIKE `B_`; (where '_' represents one unique character)
    Finds : `B1`, `Bd`, `B `
    Doesn't find : `xB`, `B`, `B12`
...WHERE `**word**` LIKE `%BASE%`; (where '%' represents zero, one or many characters)
    Finds : `DATABASE`, `BASE Jumping`
    Doesn't find : `database`, `Base`, `B A SE`
...WHERE `**age**` NOT BETWEEN...
...WHERE `**color**` NOT IN...
...WHERE `**word**` NOT LIKE...
...WHERE `**item**` IN (SELECT `**item2**` FROM `**table2**` WHERE condition)

SQL introduction

```
MariaDB [test]> SELECT * FROM person;
+--------+---------+------------+------------+
| id_per | surname | first_name | dob        |
+--------+---------+------------+------------+
|      1 | Foucart | Adrien     | 1988-04-11 |
|      2 | Dupont  | Jean       | 1970-08-24 |
|      3 | Smith   | John       | 1950-12-21 |
+--------+---------+------------+------------+
3 rows in set (0.00 sec)
```

```
MariaDB [test]> SELECT * FROM doctor;
+--------+-------+--------+
| id_doc | inami | id_per |
+--------+-------+--------+
|      1 | 12345 |      1 |
|      2 | 67890 |      3 |
+--------+-------+--------+
2 rows in set (0.00 sec)
```

```
+-------+---------+------------+
| inami | surname | first_name |
+-------+---------+------------+
| 12345 | Foucart | Adrien     |
+-------+---------+------------+
1 row in set (0.00 sec)
```

SELECT inami, surname, first_name
FROM doctor, person
WHERE inami = "12345";

```
MariaDB [test]> SELECT inami, surname, first_name FROM doctor, person WHERE inami = "12345";
+-------+---------+------------+
| inami | surname | first_name |
+-------+---------+------------+
| 12345 | Foucart | Adrien     |
| 12345 | Dupont  | Jean       |
| 12345 | Smith   | John       |
+-------+---------+------------+
3 rows in set (0.00 sec)
```

SQL introduction

```
MariaDB [test]> SELECT * FROM person;
+--------+---------+------------+------------+
| id_per | surname | first_name | dob        |
+--------+---------+------------+------------+
|      1 | Foucart | Adrien     | 1988-04-11 |
|      2 | Dupont  | Jean       | 1970-08-24 |
|      3 | Smith   | John       | 1950-12-21 |
+--------+---------+------------+------------+
3 rows in set (0.00 sec)
```

```
MariaDB [test]> SELECT * FROM doctor;
+--------+-------+--------+
| id_doc | inami | id_per |
+--------+-------+--------+
|      1 | 12345 |      1 |
|      2 | 67890 |      3 |
+--------+-------+--------+
2 rows in set (0.00 sec)
```

```
+-------+---------+------------+
| inami | surname | first_name |
+-------+---------+------------+
| 12345 | Foucart | Adrien     |
+-------+---------+------------+
1 row in set (0.00 sec)
```

```
SELECT d.inami, p.surname, p.first_name
FROM doctor d, person p
WHERE d.inami = "12345";
```

SQL introduction

```
MariaDB [test]> SELECT * FROM person;
+--------+---------+------------+------------+
| id_per | surname | first_name | dob        |
+--------+---------+------------+------------+
|      1 | Foucart | Adrien     | 1988-04-11 |
|      2 | Dupont  | Jean       | 1970-08-24 |
|      3 | Smith   | John       | 1950-12-21 |
+--------+---------+------------+------------+
3 rows in set (0.00 sec)
```

```
MariaDB [test]> SELECT * FROM doctor;
+--------+-------+--------+
| id_doc | inami | id_per |
+--------+-------+--------+
|      1 | 12345 |      1 |
|      2 | 67890 |      3 |
+--------+-------+--------+
2 rows in set (0.00 sec)
```

```
+-------+---------+------------+
| inami | surname | first_name |
+-------+---------+------------+
| 12345 | Foucart | Adrien     |
+-------+---------+------------+
1 row in set (0.00 sec)
```

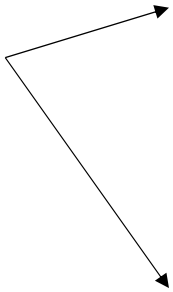
```
SELECT d.inami, p.surname, p.first_name
FROM doctor d
LEFT JOIN person p ON d.id_per = p.id_per
WHERE d.inami = "12345";
```

```
SELECT d.inami, p.surname, p.first_name
FROM doctor d
NATURAL JOIN person p
WHERE d.inami = "12345";
```

```
MariaDB [test]> SELECT d.inami, CONCAT(UPPER(p.surname), " ", p.first_name) as name
    -> FROM doctor d
    -> LEFT JOIN person p ON d.id_per = p.id_per
    -> WHERE d.inami = "12345";
+-------+----------------+
| inami | name           |
+-------+----------------+
| 12345 | FOUCART Adrien |
+-------+----------------+
1 row in set (0.00 sec)
```

SQL introduction

```
SELECT * FROM users
WHERE username = '$1'
AND password = '$2';
```

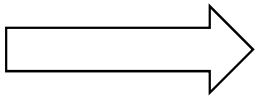


```
$1 = ' OR 1=1; /*
$2 = */--
```

```
SELECT * FROM users
WHERE username = '' OR 1=1; /*'
AND password = '*/--';
```

```
$1 = '; DROP DATABASE mydb; /*
$2 = */--
```

```
SELECT * FROM users
WHERE username = '';
DROP DATABASE mydb; /*'
AND password = '*/--';
```



SQL Injection

SQL introduction

- **Transactions** are mainly used to solve two problems:
  - What happens when two users are connected to one database at the same time?
  - What happens when one user wants to execute multiple related commands, but one of them fails?
- A transaction is a **series of commands** that we want to "protect" against those issues.

SQL introduction

```
SELECT balance FROM account WHERE id = 1; # 2000
```

```
SELECT balance FROM account WHERE id = 2; # 2000
```

```
UPDATE account SET balance = 1000 WHERE id = 1; # Withdraw 1000 euros from account 1
```

```
UPDATE account SET balance = 3000 WHERE id = 2; # Deposit 1000 euros on account 2
```

```
# If the second UPDATE fails, we have a problem.
```

SQL introduction

START TRANSACTION; # Restoration point is created for the database

UPDATE account **SET** balance = 1000 **WHERE** id = 1;

UPDATE account **SET** balance = 3000 **WHERE** id = 2;

COMMIT ; # Apply changes only if all queries succeeded

- If there is an error during the transaction, the database is restored to the "pre-transaction" state.
- The tables used for the transaction are blocked to other users until the transaction is over.

Building a (medical) software

- Software design
- Databases design
  - Organizing the data
  - Building the database
  - SQL introduction
- Java – what is it again?

Java – what is it again?

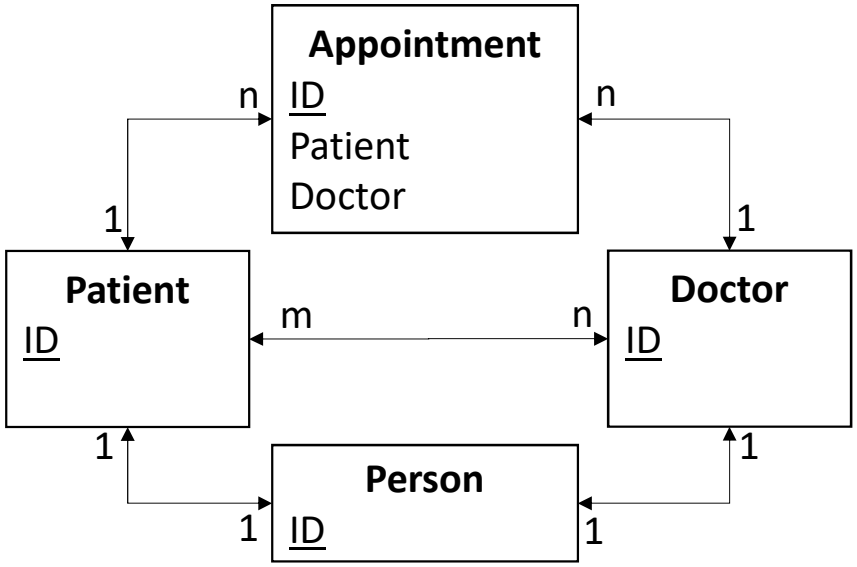
- Java is a **strictly object oriented** language.
- Java files are *precompiled* then *interpreted* by the "Java Virtual Machine". This means that Java applications are multi-platforms… as long as the JVM is installed.
- Java is good for quickly developping Graphical User Interfaces.
- Java is bad for applications with "heavy" computations.
- Java applications can be standalone or web-based (applets or servlets/web pages with an Apache Tomcat server).

Java – what is it again?

- Software design with Object-Oriented Programming (OOP) is usually very **schematic**, with emphasis on the "**actors**" and the "**interactions**" (usually written in an UML diagram)
- In an **information system** designed around a database, the "object" model is often a reflection of the "database schema".

Java – what is it again?

- For instance, this database schema could be implemented as an object model:
  - Person
    - Attributes: ID, surname, first_name…
  - Patient (*extends* Personne)
    - Attributes: social_security
  - Doctor (*extends* Personne)
    - Attributes: INAM
  - Appointment
    - Attributes : Patient, Doctor, date, time, place…

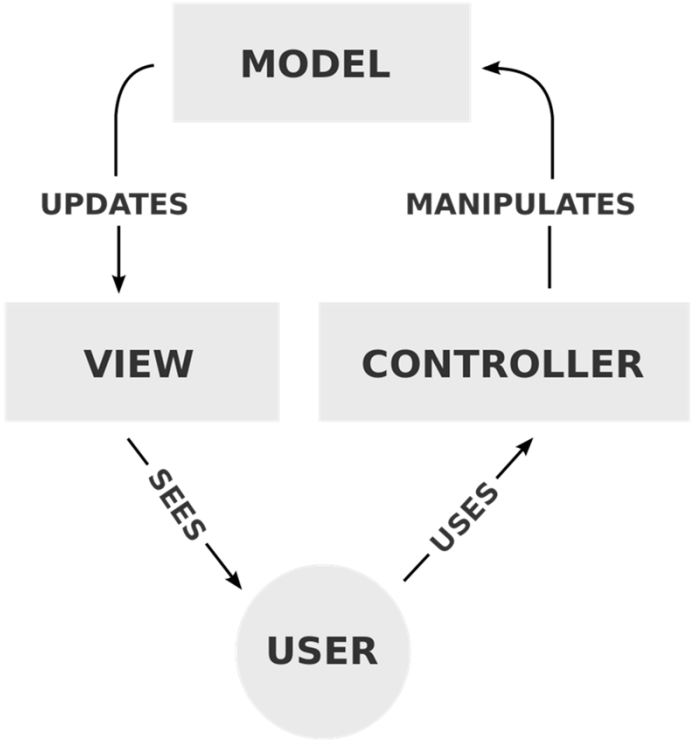


Java – what is it again?

```
public class Appointment {

          private int id_app;
          private Datetime date_time;
          private Patient patient;
          private Doctor doctor;

          public Appointment(int id, Datetime dt, Patient p, Doctor d){

                    this.id_app = id;
                    this.date_time = dt;
                    this.patient = p;
                    this.doctor = d;

          }

          public int getId(){

                    return this.id_app;

          }

          …

}
```

Java – what is it again?

- Those classes, in the "MVC" design pattern, would be the "model" of our application.
- We could then add the graphical interface (the "view"), as well as "controller" classes to handle actions.



Some final thoughts…

- During the labs, we will use the **NetBeans** IDE, which includes a design tool to create graphical user interfaces.
- The development work will generally be to correctly **identify** and **understand** the different elements needed for the application, and to **link them together**.
- Google is the developper's best friend. Learn to **find good keywords**, to **identify good sources** (e.g. StackOverflow) and **bad sources** (e.g. old programming forums with obsolete/no answers).