OO Design with UML

Chouki Tibermacine

Chouki.Tibermacine@umontpellier.fr

Outline

1. UML as an OO Modeling Language

2. Use Case Diagrams

3. Class Diagrams

Part #1 : Introduction

Outline

1. UML as an OO Modeling Language

2. Use Case Diagrams

3. Class Diagrams

UML in Software Design

- UML (*Unified Modeling Language*) is an industrial standard for software modeling (data, business logic, workflows, ...)
- UML = Unified language for modeling software in terms of (but not limited to) objects (some models do not require the use of objects)
- “Unified” because in the past, there were many notations for modeling different viewpoints on software systems

What is UML exactly ?

Software Modeling Language

- Result of research works in software engineering conducted within large industrial groups
- Used in *Software Analysis* and *Design*, but also in *Maintenance*

UML is not a Method

- It is a language and not a method
- Language designed to be used with any method
- Language developed by Rational (acquired by IBM) with a method named RUP (*Rational Unified Process*)

Some History about UML

- UML is the result of merging three methods and languages developed separately :
  - OMT, a technique developed by James Rumbaugh (General Electric → Rational in 1994)
  - Booch, method developed by Grady Booch (Rational)
  - OOSE, method developed by Ivar Jackobson (Ericsson → Rational in 1995)
- The first version of the language was specified in 1996, standardized by the OMG (*Object Management Group*) in 1997 and by ISO in 2005
- Two major versions : UML 1 (0.9 → 1.0) and UML 2 (2005)
- Current version : 2.5.1 (spec published in déc. 2017)

UML and Modeling in IT

- UML enables to model different viewpoints on an information system :
  - data and processes
  - static (structural) view and dynamic (behavioral) view
- In practice (in professional teams), UML is not used entirely [1]
- Only a small part of it is used (explained later)

[1] UML in practice. Peter Marian. In Proceedings of ICSE'13.

Do not mix up UML *Models* and *Diagrams*

First, what is a model ?

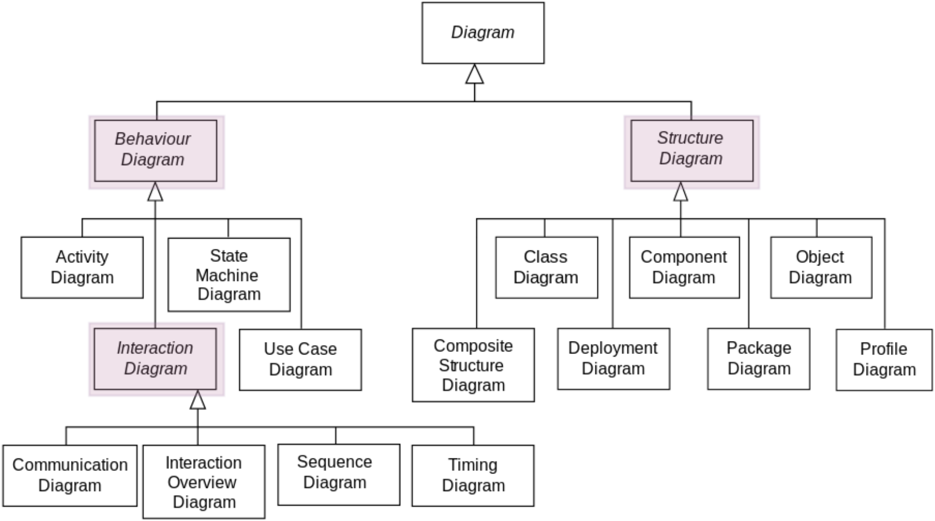
Abstract representation of the relations between the characteristic parameters of a phenomenon or a process [2]

And in UML ?

- A model is a set of software modeling elements (classes, objects, messages, use cases, ...) which are documented (named and characterized)
- A diagram is a graphical partial representation (a possible view) of a model of a system
- A diagram can be deleted without any impact on the model (just the graphical view disappears)
- In practice, we build a diagram ⇒ model elements are created automatically (and are reusable in other diagrams)
- This is how UML editors work

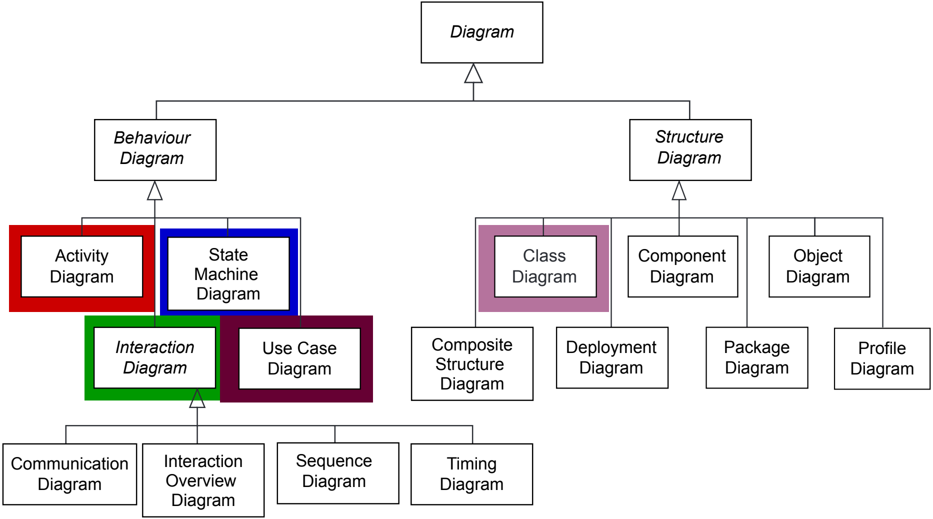
[2] Scientific Model – Larousse

Different Kinds of Diagrams [3]



[3] https ://en.wikipedia.org/wiki/Unified_Modeling_Language

Which Diagrams in this Course ?



Why these Diagrams ?

- Use Case Diagrams : Modeling in the early stages (Analysis) the basic functionality provided by the software system and which actors interact with it

- Class Diagrams : Modeling data schemas and the structure of business processes (classes with their members and relationships)

- Activity Diagrams : Detailing business processes by showing control and data flows

- State Diagrams : Modeling the evolution of the states of the software system and its objects during the execution of processes

- Interaction Diagrams : Modeling the scheduling of message passing and chronology of interactions

Organization

Part 2

- **Use Case Diagrams :** Modeling the basic functionality provided by the software system and which actors interact with it
- **Class Diagrams :** Modeling data schemas and the structure of business processes (classes with their members and relationships)

Part 3

- **Interaction Diagrams :** Modeling the scheduling of message passing and chronology of interactions

Part 4

- **Activity Diagrams :** Detailing business processes & their control and data flows
- **State Diagrams :** Modeling the evolution of the states of the software system and its data during the execution of processes

Part #2 : Use Case & Class Diagrams

Outline

1. UML as an OO Modeling Language

2. Use Case Diagrams

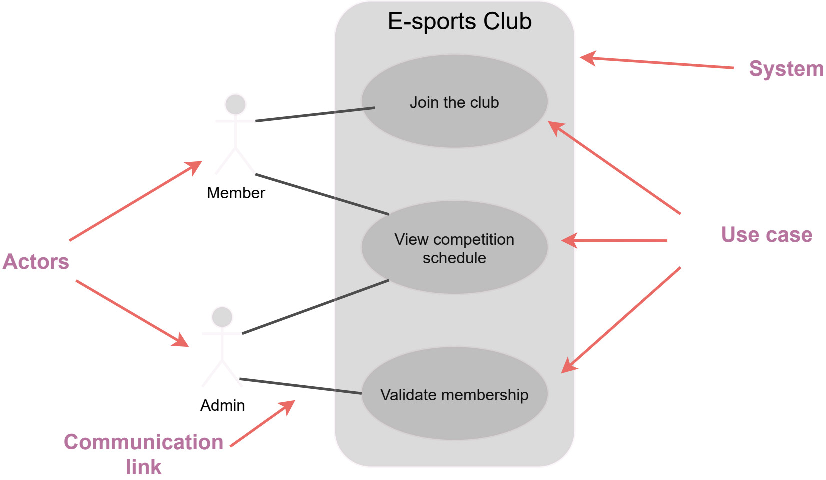
3. Class Diagrams

What is a Use Case Diagram ?

- It is a UML diagram which depicts a general overview on the functionality provided by a software system
- It enables to answer the following question :

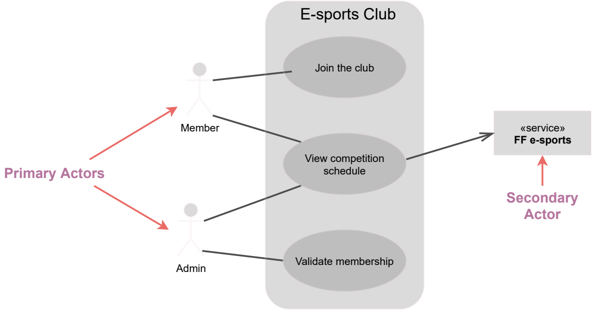
  “Who can do what on the software system ?”
- It helps to represent in a graphical way the functions (**use cases**) of the software system from a user point of view
- It enables to indicate also what are the external entities (**actors**) which interact with the system
- We define it to **capture the needs/requirements** of a client during the development of a software system (can be defined as a refinement of *user stories*)

Example of a use case diagram



Primary and Secondary Actors

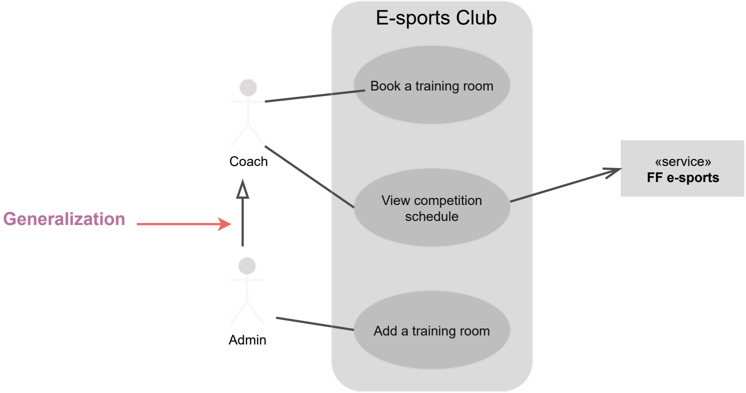
- Primary Actor : a person or an external system which triggers the execution of a use case



- Secondary Actor : a person or an external system which helps in the execution of a use case, or which receives data from the system

Relations between actors

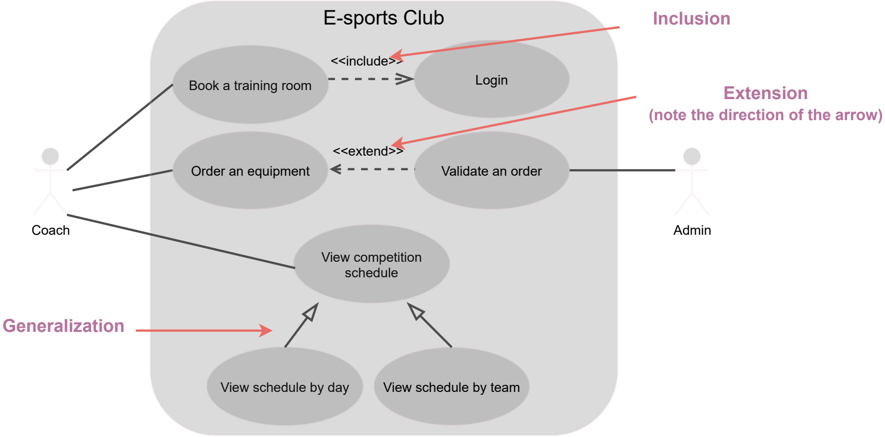
- Actors may be linked by generalization (inheritance) relations : **an actor inherits** thus all the capabilities of its “parent” to execute other use cases



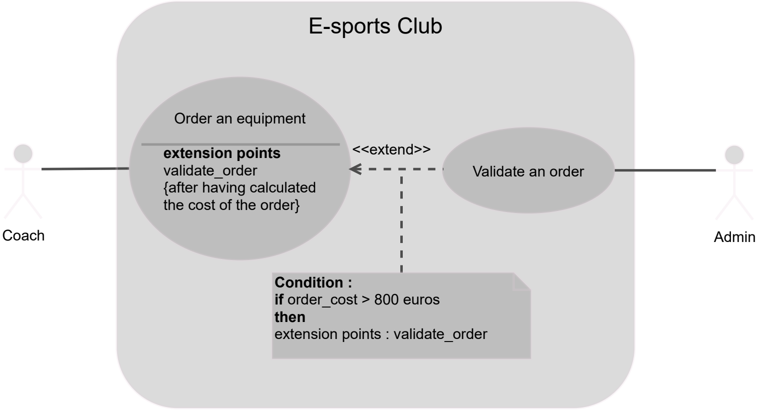
Relations between use cases

1. Relation of type “include” : the execution of the source use case includes mandatorily the execution of the target use case
2. Relation of type “extend” : the execution of the source use case **may** follow the execution of the target use case (*it is an option*)
3. Relation of type generalization (inheritance) : a use case is a particular case (a specialization) of its parent use case

Examples of relations between use cases



Extension points in use cases



Some good practices

- **Actors** are **roles** played by **categories of human users** and not precise users
- They can also be other systems which are outside the borders of the software system (and not subsystems of the system)
- Actors interact **directly** with the software system
- Use cases must be at the same level of abstraction (not too many/detailed)
- There should be no redundancy in functionality modeled by use cases (orthogonal UCs)
- The name of a use case = infinitive verb = action of the actor on the system (and not the inverse)

UC Diagram vs User Stories

- Both identify users and describe goals, but they serve different purposes
- User Stories are centered on the result and the benefit
- Use Cases can be more granular, and describe how the system will act
- User Stories contain, with user role, goal and acceptance criteria (lack of largest goal)
- A detailed Use Case template may contain many more other elements (UC name, precondition, postcondition, basic path, alternative paths and exception paths)

https://www.visual-paradigm.com/guide/agile-software-development/user-story-vs-use-case/

Exercice

Use PlantUML (`https://plantuml.com/fr/use-case-diagram`) to define :

- a use case diagram for the restaurants website
- with at least two actors and 6-7 use cases
- add some relations of type `extend` and `include` between use cases

The PlantUML editor is available here : `www.plantuml.com/`

Outline

1. UML as an OO Modeling Language
2. Use Case Diagrams
3. Class Diagrams

Roles of class diagram in software development

What is a class diagram ?

A graphical representation of classes, their characteristics (members) and their relationships

Why we define class diagrams ?

1. For modeling persistent data schemas
2. For designing the code of an application, starting from use cases and descriptions of scenarios of UCs
   - Code is defined in terms of `descriptors` of objects that realize the use cases

What is an object and what is a class ?

An object

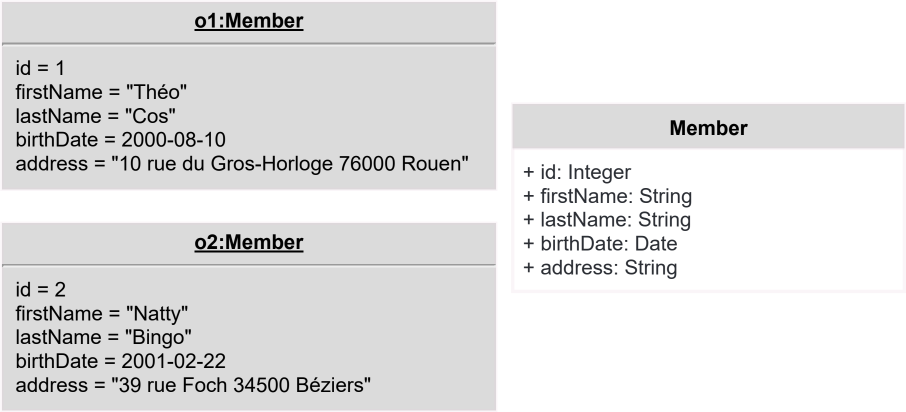
An entity of the business and technical domains having the following characteristics :

- **a state :** a set of slots (pairs of attribute-name and its value)
- **a behavior :** a set of operations (signatures of methods)
  ⇒ the know-how of the object

A class

A descriptor for objects having the same characteristics

Examples of objects and their class



o1:Member

id = 1
firstName = "Théo"
lastName = "Cos"
birthDate = 2000-08-10
address = "10 rue du Gros-Horloge 76000 Rouen"

o2:Member

id = 2
firstName = "Natty"
lastName = "Bingo"
birthDate = 2001-02-22
address = "39 rue Foch 34500 Béziers"

Member

+ id: Integer
+ firstName: String
+ lastName: String
+ birthDate: Date
+ address: String

- No operations here

Description of attributes in classes

Syntax

$< visibility >< attr.\_name >:< attr.\_type > [=< initial\_value >][\{props\}]$

Visibility

+ public, - private, # protected and $\sim$ package-private

Examples

+ name : String

- isActivated : boolean = false

Description of operations in classes

Syntax

$< visibility >< op\_name > \mathbf{(}< params\_list >\mathbf{)}[:< return\_type >]$

Visibility : the same as for attributes

List of parameters

Syntax :

$< in|out|inout >< param\_name >:< param\_type >$

Examples

+ getName() : String

+ setName(in name : String)

Relations between classes

Inheritance or Generalization

- See next OOP course on Inheritance

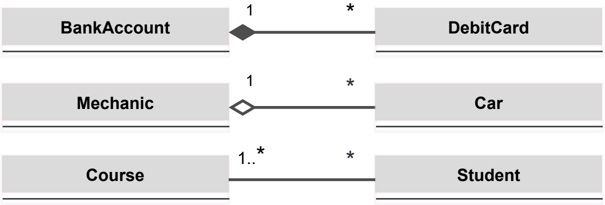
Dependency

- A class A requires another class B in its specification or implementation (A creates objects of B, for ex.)
- Not necessarily useful in modeling data schemas

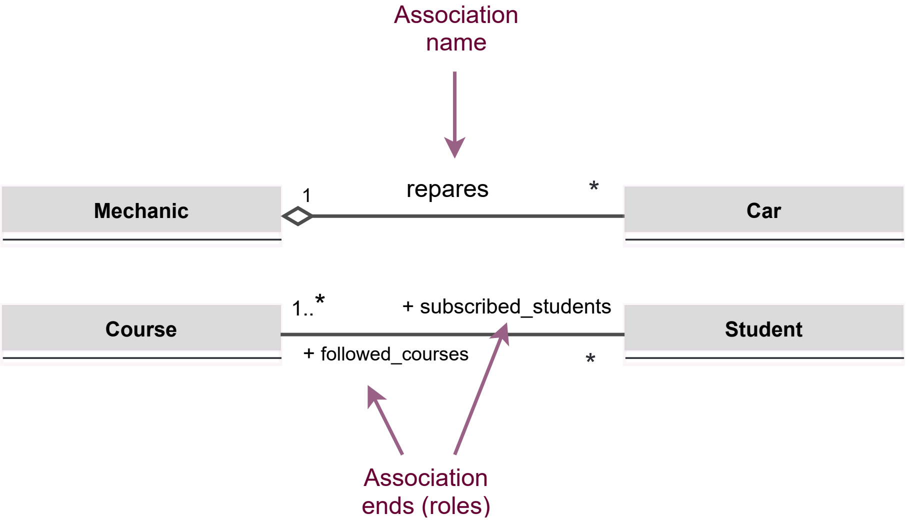
Relations between classes -ctd-

Association (simple, aggregation ou composition

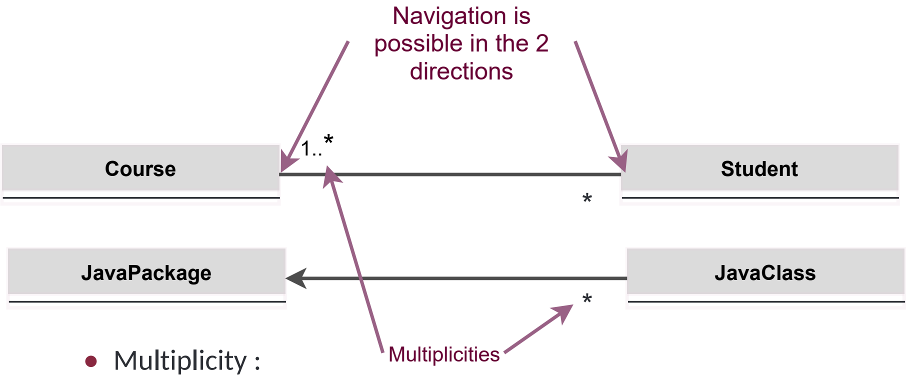
- An association describes a set of **links** between objects
- A structural relation between objects
- Composition : strong relation (linking the lifecyles of objects)
- Aggregation : relation of sharing (shared object)
- Simple association : the least constraining (most general) relation



Names and roles (association ends) for relations



Navigability and multiplicity

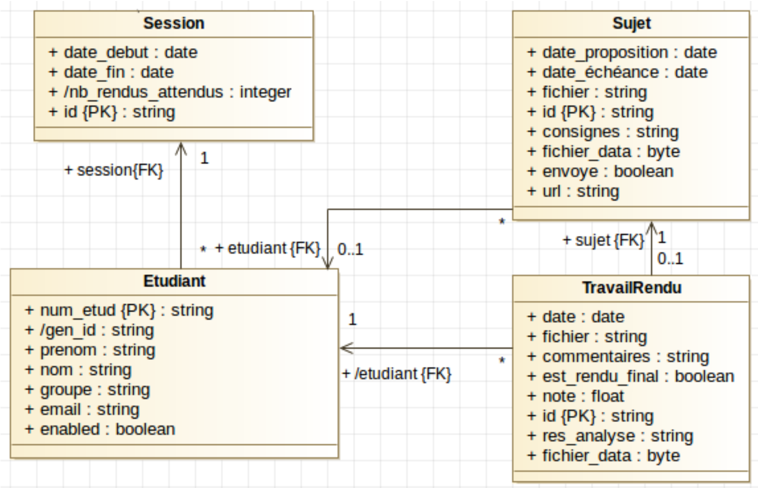


- Multiplicity :
  - n : n is a number or *
  - n..m : n and m are numbers or 1..*
  - Default value (when nothing is indicated), it is 1

Specific case for DB schemas

- A database schema can be modeled by a class diagram
- A table = A class
- The name of a column in a table = an attribute
- A record in the table = an object
- The properties of columns (primary key PK, not null value, ...) can be defined as properties of attributes
- A foreign key FK can be defined as role of an association

Example of class diagram (DB of Plage App v1)



Session

+ date_debut : date
+ date_fin : date
+ /nb_rendus_attendus : integer
+ id {PK} : string

Sujet

+ date_proposition : date
+ date_échéance : date
+ fichier : string
+ id {PK} : string
+ consignes : string
+ fichier_data : byte
+ envoye : boolean
+ url : string

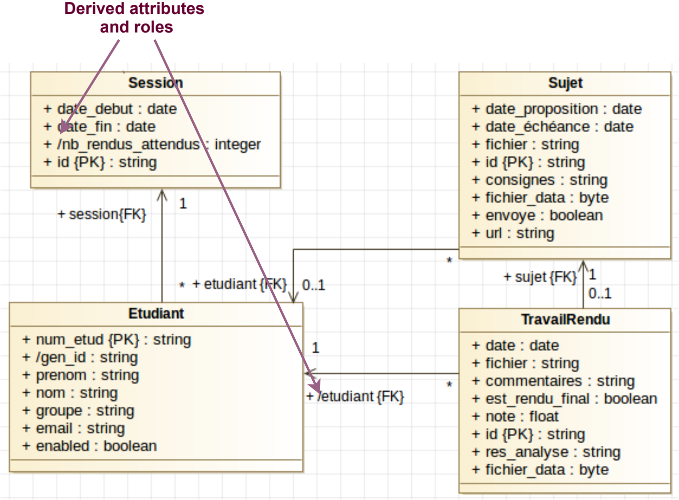
Etudiant

+ num_etud {PK} : string
+ /gen_id : string
+ prenom : string
+ nom : string
+ groupe : string
+ email : string
+ enabled : boolean

TravailRendu

+ date : date
+ fichier : string
+ commentaires : string
+ est_rendu_final : boolean
+ note : float
+ id {PK} : string
+ res_analyse : string
+ fichier_data : byte

Example of class diagram -ctd-



Derived attributes and roles

Session

+ date_debut : date
+ date_fin : date
+ /nb_rendus_attendus : integer
+ id {PK} : string

Sujet

+ date_proposition : date
+ date_échéance : date
+ fichier : string
+ id {PK} : string
+ consignes : string
+ fichier_data : byte
+ envoye : boolean
+ url : string

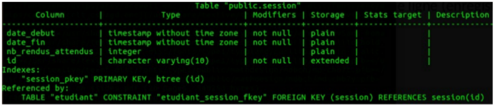
Etudiant

+ num_etud {PK} : string
+ /gen_id : string
+ prenom : string
+ nom : string
+ groupe : string
+ email : string
+ enabled : boolean

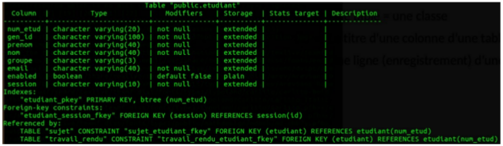
TravailRendu

+ date : date
+ fichier : string
+ commentaires : string
+ est_rendu_final : boolean
+ note : float
+ id {PK} : string
+ res_analyse : string
+ fichier_data : byte

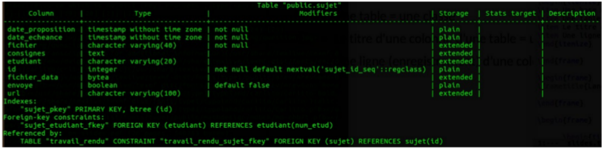
Example of tables



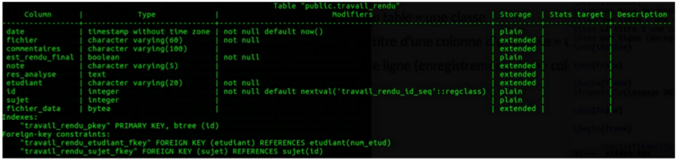
```
                                Table "public.session"
      Column       |            Type             | Modifiers | Storage  | Stats target | Description
-------------------+-----------------------------+-----------+----------+--------------+-------------
 date_debut        | timestamp without time zone | not null  | plain    |              |
 date_fin          | timestamp without time zone | not null  | plain    |              |
 nb_rendus_attendus | integer                    |           | plain    |              |
 id                | character varying(10)       | not null  | extended |              |
Indexes:
    "session_pkey" PRIMARY KEY, btree (id)
Referenced by:
    TABLE "etudiant" CONSTRAINT "etudiant_session_fkey" FOREIGN KEY (session) REFERENCES session(id)
```



```
                              Table "public.etudiant"
  Column  |          Type          |   Modifiers   | Storage  | Stats target | Description
----------+------------------------+---------------+----------+--------------+-------------
 num_etud | character varying(20)  | not null      | extended |              |
 gen_id   | character varying(100) | not null      | extended |              |
 prenom   | character varying(40)  | not null      | extended |              |
 nom      | character varying(40)  | not null      | extended |              |
 groupe   | character varying(3)   |               | extended |              |
 email    | character varying(40)  | not null      | extended |              |
 enabled  | boolean                | default false | plain    |              |
 session  | character varying(10)  | not null      | extended |              |
Indexes:
    "etudiant_pkey" PRIMARY KEY, btree (num_etud)
Foreign-key constraints:
    "etudiant_session_fkey" FOREIGN KEY (session) REFERENCES session(id)
Referenced by:
    TABLE "sujet" CONSTRAINT "sujet_etudiant_fkey" FOREIGN KEY (etudiant) REFERENCES etudiant(num_etud)
    TABLE "travail_rendu" CONSTRAINT "travail_rendu_etudiant_fkey" FOREIGN KEY (etudiant) REFERENCES etudiant(num_etud)
```

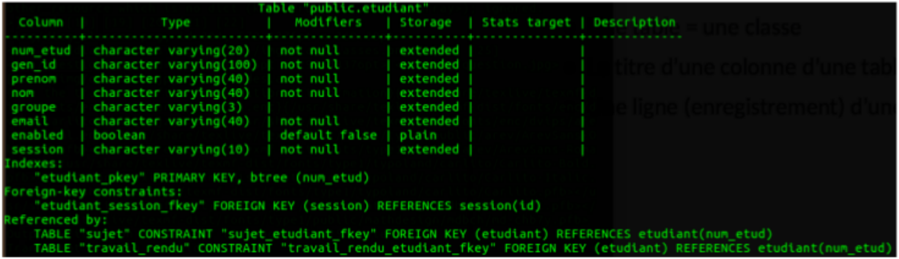


```
                                                 Table "public.sujet"
      Column      |            Type             |                       Modifiers                        | Storage  | Stats target | Description
------------------+-----------------------------+--------------------------------------------------------+----------+--------------+-------------
 date_proposition | timestamp without time zone | not null                                               | plain    |              |
 date_echeance    | timestamp without time zone | not null                                               | plain    |              |
 fichier          | character varying(40)       | not null                                               | extended |              |
 consignes        | text                        |                                                        | extended |              |
 etudiant         | character varying(20)       |                                                        | extended |              |
 id               | integer                     | not null default nextval('sujet_id_seq'::regclass)     | plain    |              |
 fichier_data     | bytea                       |                                                        | extended |              |
 envoye           | boolean                     | default false                                          | plain    |              |
 url              | character varying(100)      |                                                        | extended |              |
Indexes:
    "sujet_pkey" PRIMARY KEY, btree (id)
Foreign-key constraints:
    "sujet_etudiant_fkey" FOREIGN KEY (etudiant) REFERENCES etudiant(num_etud)
Referenced by:
    TABLE "travail_rendu" CONSTRAINT "travail_rendu_sujet_fkey" FOREIGN KEY (sujet) REFERENCES sujet(id)
```



```
                                                 Table "public.travail_rendu"
     Column      |            Type             |                         Modifiers                          | Storage  | Stats target | Description
-----------------+-----------------------------+------------------------------------------------------------+----------+--------------+-------------
 date            | timestamp without time zone | not null default now()                                     | plain    |              |
 fichier         | character varying(60)       | not null                                                   | extended |              |
 commentaires    | character varying(100)      |                                                            | extended |              |
 est_rendu_final | boolean                     | not null                                                   | plain    |              |
 note            | character varying(5)        |                                                            | extended |              |
 res_analyse     | text                        |                                                            | extended |              |
 etudiant        | character varying(20)       | not null                                                   | extended |              |
 id              | integer                     | not null default nextval('travail_rendu_id_seq'::regclass) | plain    |              |
 sujet           | integer                     |                                                            | plain    |              |
 fichier_data    | bytea                       |                                                            | extended |              |
Indexes:
    "travail_rendu_pkey" PRIMARY KEY, btree (id)
Foreign-key constraints:
    "travail_rendu_etudiant_fkey" FOREIGN KEY (etudiant) REFERENCES etudiant(num_etud)
    "travail_rendu_sujet_fkey" FOREIGN KEY (sujet) REFERENCES sujet(id)
```

Table “etudiant”

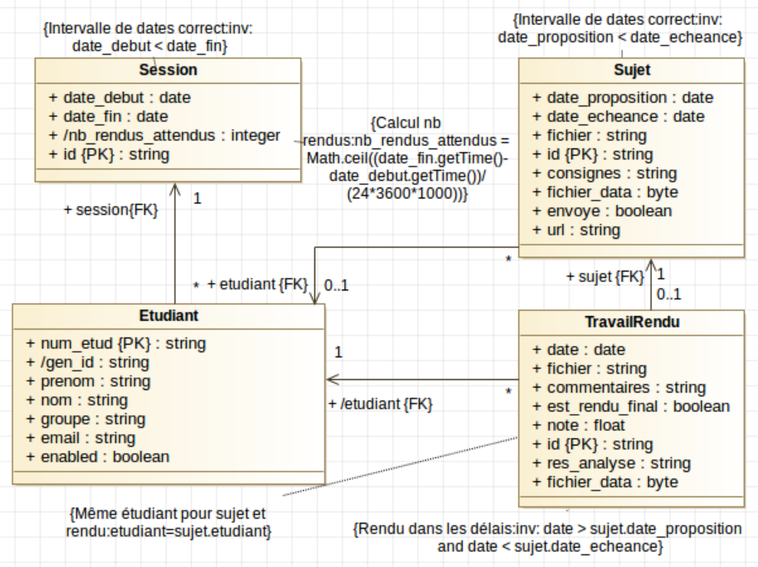


```
                              Table "public.etudiant"
  Column  |          Type          |   Modifiers   | Storage  | Stats target | Description
----------+------------------------+---------------+----------+--------------+-------------
 num_etud | character varying(20)  | not null      | extended |              |
 gen_id   | character varying(100) | not null      | extended |              |
 prenom   | character varying(40)  | not null      | extended |              |
 nom      | character varying(40)  | not null      | extended |              |
 groupe   | character varying(3)   |               | extended |              |
 email    | character varying(40)  | not null      | extended |              |
 enabled  | boolean                | default false | plain    |              |
 session  | character varying(10)  | not null      | extended |              |
Indexes:
    "etudiant_pkey" PRIMARY KEY, btree (num_etud)
Foreign-key constraints:
    "etudiant_session_fkey" FOREIGN KEY (session) REFERENCES session(id)
Referenced by:
    TABLE "sujet" CONSTRAINT "sujet_etudiant_fkey" FOREIGN KEY (etudiant) REFERENCES etudiant(num_etud)
    TABLE "travail_rendu" CONSTRAINT "travail_rendu_etudiant_fkey" FOREIGN KEY (etudiant) REFERENCES etudiant(num_etud)
```

OCL language for defining constraints

- Conditions on values of attributes and on associations that cannot be expressed directly with UML
- OCL (*Object Constraint Language*) : standard of the OMG
- OCL constraint = predicate (expression in 1st order logic)
- Language very easy to learn
- Powerful langage for refining the semantics of UML models

Examples of OCL constraints



{Intervalle de dates correct:inv:
date_debut < date_fin}

{Intervalle de dates correct:inv:
date_proposition < date_echeance}

{Calcul nb
rendus:nb_rendus_attendus =
Math.ceil((date_fin.getTime()-
date_debut.getTime())/
(24*3600*1000))}

{Même étudiant pour sujet et
rendu:etudiant=sujet.etudiant}

{Rendu dans les délais:inv: date > sujet.date_proposition
and date < sujet.date_echeance}

Tools

Editors and Tools

- General purpose editors : diagrams.net (draw.io), Dia, ...
- Dedicated environments :
  - Proprietary : Modelio from Modeliosoft/Softeam (`https://www.modelio.org/`), Visual Paradigm (`https://www.visual-paradigm.com/`), Rational Software Architect from IBM, ...
  - Free : Papyrus from CEA LIST (Commissariat à l'Énergie atomique et aux Énergies alternatives), Umbrello, ArgoUML (Tigris.org), ...
  - Textual Syntax : PlantUML (`https://plantuml.com/`)

Features

- Graphical/Textual editing and model validation
- Model transformation, code generation, ...

Exercice

Use PlantUML (`https://plantuml.com/fr/class-diagram`) to define :

- A class diagram for the Labyrinth Application
- with all possible relations between classes and specify cardinalities and aggregation types (composition, ...)
- Drop your (UC and class) diagrams (source and PNG)in Moodle

The PlantUML editor is available here : `www.plantuml.com/`