



UNIVERSITÀ DI PISA

DEPARTMENT OF INFORMATION ENGINEERING

# Information Systems Task 1 Documentation

STUDENTS:

ADRIANO BOTTI

ANTONIO LE CALDARE

FRANCESCO MEROLA

GIACOMO PONZIANI

# Contents

<b>List of Figures</b>	<b>2</b>
<b>Application Specifications</b>	<b>2</b>
<b>Requirements and Use Cases</b>	<b>3</b>
Functional Requirements . . . . .	4
Non-Functional Requirements . . . . .	4
Use Case Diagram . . . . .	5
<b>UML Class and Entity-Relationship Diagrams</b>	<b>6</b>
UML Class Diagram . . . . .	6
ER Vocabulary . . . . .	6
ER Diagram . . . . .	7
Database Implementation . . . . .	8
<b>Key-Value Feasibility Study</b>	<b>8</b>
<b>User's Manual</b>	<b>9</b>

# List of Figures

1	Use Case Diagram . . . . .	5
2	UML Class Diagram . . . . .	6
3	Entity - Relationship Diagram . . . . .	8
4	MySQL Schema . . . . .	8
5	User Interface when the application starts . . . . .	10
6	Example of Add Exam . . . . .	11
7	Example of Add Exam dialog . . . . .	12
8	Example of Add Grade . . . . .	12
9	Example of Add Grade dialog . . . . .	13
10	Example of Register to Exam . . . . .	13
11	Example of Deregister to Exam . . . . .	14
12	Example of See Grades . . . . .	15

# Application Specifications

The goal of the application that we implemented is to provide a way for both students and professors to manage the registration process for exams. More specifically, we want the application to exert the following functionalities:

- For Students:
  1. Check past exams results
  2. Register to an exam date
  3. Delete an exam registration
- For Professors:
  1. Add grades to an exam
  2. Create a new exam date

# Requirements and Use Cases

## Functional Requirements

The Professor:

1. shall be able to insert an exam, associated with a course he holds, in a date of choice
2. shall not be able to insert an exam for a date precedent to the current date
3. shall be able to insert the corresponding grade for a student in his registration for that exam.
4. shall insert all grades in the exact date of the exam

The Student:

1. shall be able to check the results of past exams
2. shall be able to register to an exam not yet took
3. shall not be able to register to an exam after the exam date.
4. shall be able to register to more successive exams for the same course
5. If the student registered to successive exams for the same course he just got a mark for, then those future registrations shall be deleted
6. shall be able to deregister from an exam he was previously registered to
7. shall not be able to deregister from an exam already took
8. shall not be able to deregister from an exam after the exam date.

## Non-Functional Requirements

For the application we identified Consistency and Availability as the two most important non-functional requirements.

# Use Case Diagram

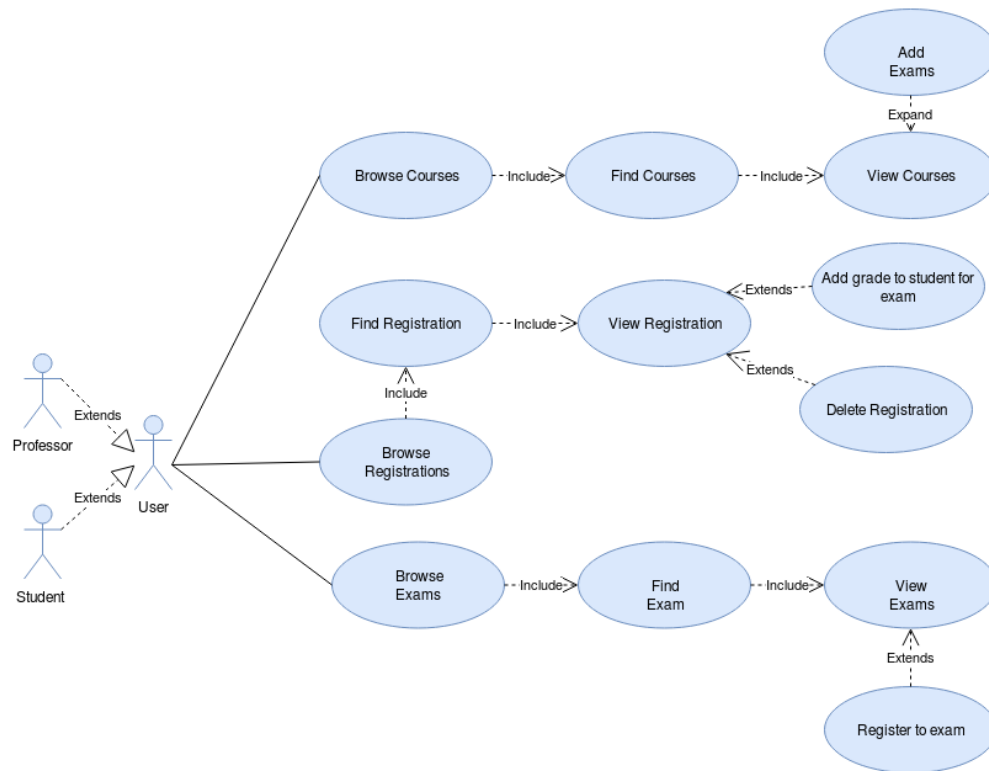


Figure 1: Use Case Diagram

- **Add Exam** : This operation can only be performed by a Professor.  
He/She will be able to browse his/her tenured courses, select the one to add a new exam to and then add it.
- **Add Grade** : This operation can only be performed by a Professor.  
He/She will be able to browse the registrations for his/her tenured courses, select the registration to add a grade to and then add it.
- **Register to Exam** : This operation can only be performed by a Student.  
He/She will be able to browse all the exams to which is possible to register, select one and perform the registration.
- **Deregister to Exam** : This operation can only be performed by a Student.  
He/She will be able to browse his/her active registrations, select one and deregister from it.
- **See Grades** : This operation can only be performed by a Student.  
He/She will be able to browse his/her registrations and show only the one with a sufficient.

For more detail and a step-by-step description of the different scenarios, see chapter *User's Manual*.

# UML Class and Entity-Relationship Diagrams

## UML Class Diagram

The UML Class Diagram shows the main objects involved in our application, with the respective logical relationships. From this model we derived an Entity-Relationship Diagram, where the

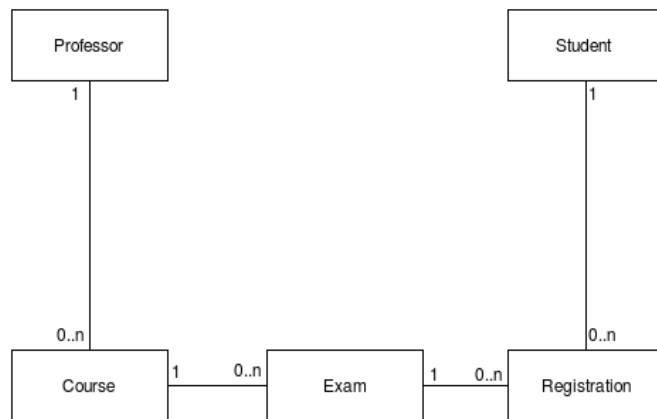


Figure 2: UML Class Diagram

dependencies are more deeply identified. Finally we will show the final database schema implemented on the MySQL RDBMS.

## ER Diagram

### Names definition

Here we define in detail the terms for the main entites and relationships we will use in the following:

- *Student*  
A student is an entity which is able to perform the operations already defined in the requirements. He's an actor for our application.
- *Professor*  
A professor is an entity which is able to perform the operations already defined in the requirements. He's an actor for our application.
- *Course*  
A course is held by one and only one professor. The course object only includes information about its name, cfu and the holding professor. It holds no information about when exams for that course will take place.
- *Exam*  
An exam represents the actual date of the examination for a course.
- *Exam Result* An exam result relates an exam to all the students who registered to that exam, adding the information of the grade, if meaningful.

## Entities

Entity	Description	Attributes
Student	Holds all the information related to the students	<ul style="list-style-type: none"><li>• <u>id</u></li><li>• <i>name</i></li><li>• <i>surname</i></li></ul>
Professor	Holds all the information related to the professors	<ul style="list-style-type: none"><li>• <u>id</u></li><li>• <i>name</i></li><li>• <i>surname</i></li></ul>
Course	Holds the information related to the courses	<ul style="list-style-type: none"><li>• <u>id</u></li><li>• <i>name</i></li><li>• <i>cfu</i></li><li>• <i>professor</i></li></ul>
Exam	Holds all the new and past exams	<ul style="list-style-type: none"><li>• <u>course(ext)</u></li><li>• <u>date</u></li></ul>

## Relationships

Relationship	Description	Participants	Attributes
Teaching	Links Professors to their held courses	<ul style="list-style-type: none"><li>• Professor(1,N)</li><li>• Course(1,1)</li></ul>	
Exam Result	Links students to exams, with the respective grade	<ul style="list-style-type: none"><li>• Student(0,N)</li><li>• Exam(0,N)</li></ul>	<ul style="list-style-type: none"><li>• <i>grade</i></li></ul>
Exam Date Creation	Links the courses to the exams through a date	<ul style="list-style-type: none"><li>• Course(0,N)</li><li>• Exam(1,1)</li></ul>	



## ER Diagram

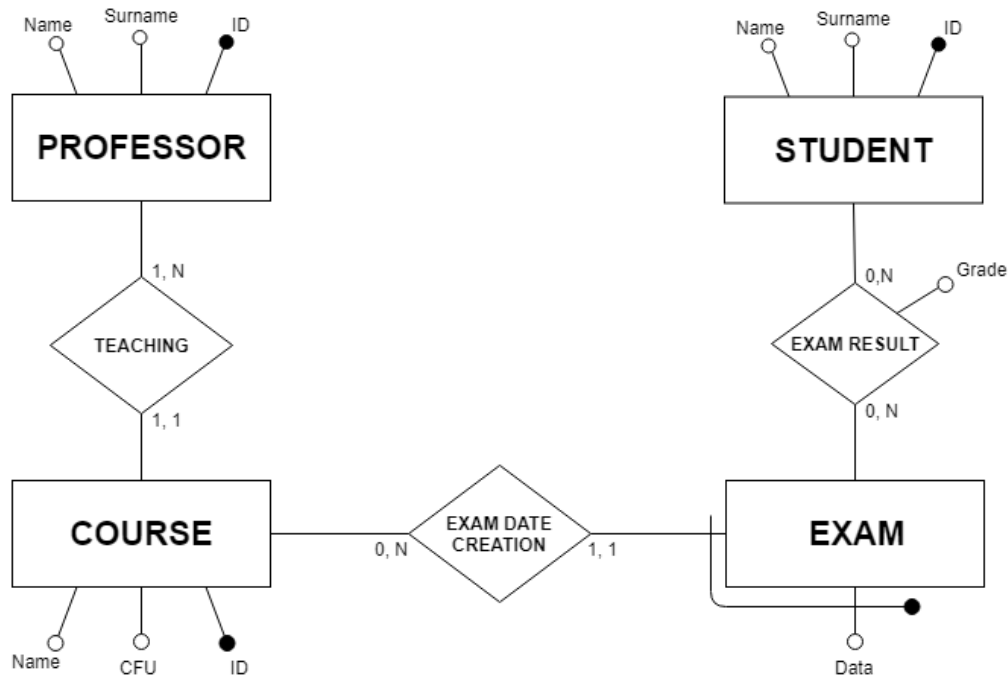


Figure 3: Entity - Relationship Diagram

## Database Implementation

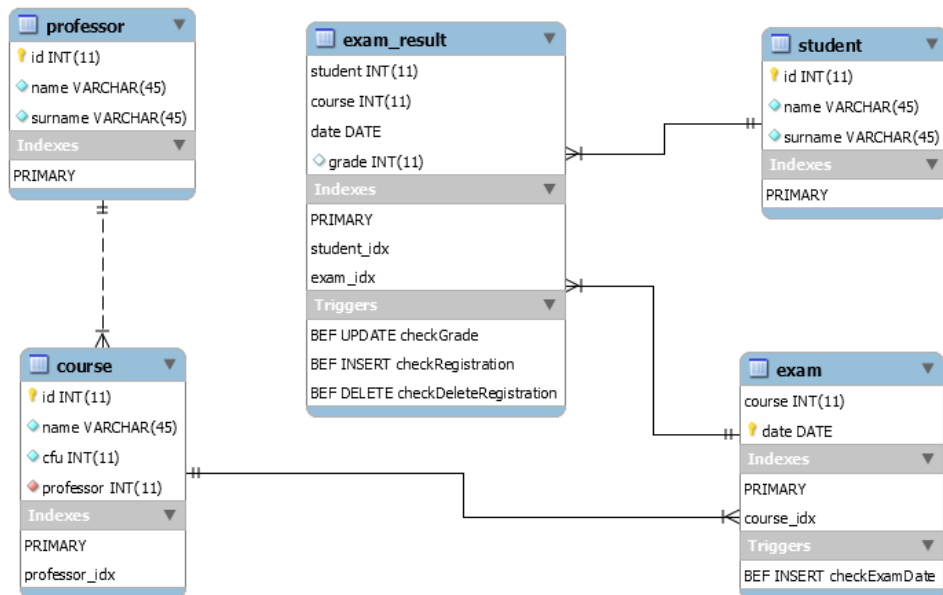


Figure 4: MySQL Schema

# Key-Value Feasibility Study

The first version of the application stores all its data in a relational database managed by the MySQL RDBMS. The main non-functional requirements are availability and consistency as they are both guaranteed by the relational database since the application handles a limited amount of data. Availability may be no more guaranteed if the number of users grows large, i.e. the number of students. The system may respond slower when too many users are active at the same time since in this scenario the relational database has to perform a lot of complex operations to fulfill their demands resulting in not negligible waiting times. Operations that work with exam registrations are the most expensive and time consuming as they require to perform a join on multiple tables of the database. Reducing this kind of operations may be a possible solution to improve the system performance: it may be convenient to use a key-value database in parallel with the relational one and spread different operations between them.

The relational database still stores all the business data and the key-value database stores only data relative to registrations. The two copies of registrations data make the storage system redundant but consistency is always guaranteed by the fact that write operations are committed on both the databases or canceled. When a write operation occurs both the databases update consistently their data and if one of two can not commit the update the other rolls back and it is so possible to perform a write operation if and only if both databases are on. If a read operation regards single-table data it is redirected towards the relational database, if instead it regards registrations it is redirected towards the key-value database. In the latter case it may happen that the key-value database is unavailable, in this case the read operation is redirected towards the relational database.

The key is composed of:

- id of the course of the exam;
- date of the exam;
- id of the professor of the course;
- id of the student.

The values, stored separately, are:

- name of the student;
- surname of the student;
- name of the professor;
- surname of the professor;
- name of the course;
- number of CFU of the course;
- grade assigned by the professor.

A key-value database is so used to take advantage of its simplicity and velocity to speed up read operations on registrations data.

# User's Manual

The application starts and shows a minimal user interface, fig. 5, composed of:

Task0

ID Utente: 1

Role: Student

What do you want to do?: See Grades

CONFIRM

Course	Date	Action
Nessun contenuto nella tabella		

Figure 5: User Interface when the application starts

- *UserID* field, where the user specifies his id number;
- *Role* choice box, with the option:
  1. Professor, in case the user is a Professor;
  2. Student, in case the user is a Student.
- *What do you want to do?* choice box, where the user specifies the action he wants to perform. The options change according to the role chosen. Professor can select among:
  1. *Add Exam* if he wants to add an exam;
  2. *Add Grade* if he wants to add a grade.A Student can select among:
  1. *Register to Exam* if he wants to register to an exam;
  2. *Deregister to Exam* if he wants to deregister to an exam;
  3. *See Grades* if he wants to see the grades he got.
- *Confirm* button;
- A table showing the results of the selected operation. The layout of the table can change according depending on the selected operation.

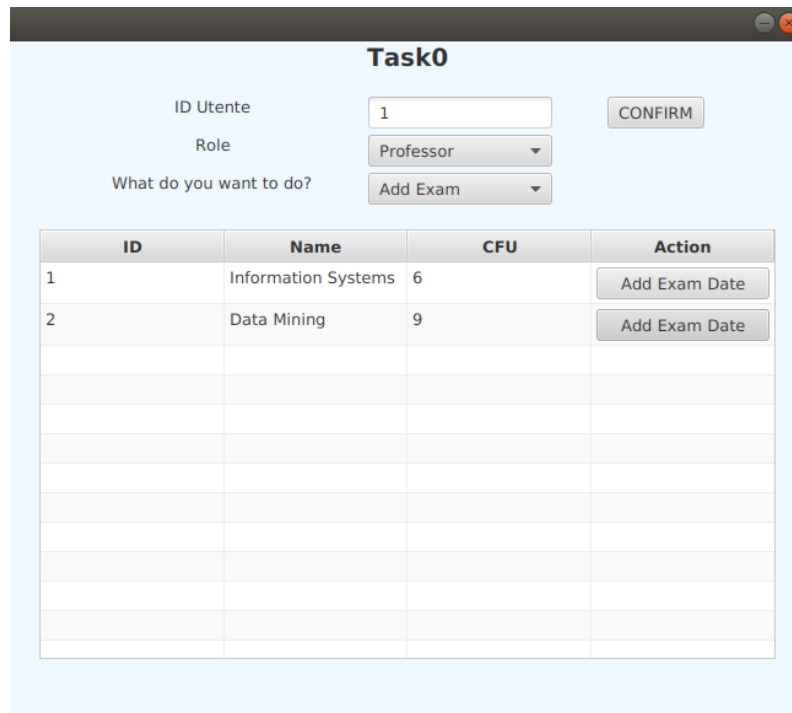
# Professor

A Professor:

1. Inserts his ID number into the *UserID* field;
2. Selects *Professor* in the *Role* choice box;
3. Selects the action he wants to perform;
4. Pushes the *Confirm* button.

## Add Exam

If the Professor select the *Add Exam* option, the table, fig. 6, is populated with a list of all the courses he is currently holding. Each element of the list is composed of:



The screenshot shows a web application window titled "Task0". It contains a form with the following fields and buttons:

- ID Utente:** A text input field containing the value "1".
- Role:** A dropdown menu with "Professor" selected.
- What do you want to do?:** A dropdown menu with "Add Exam" selected.
- CONFIRM:** A button to the right of the form.

Below the form is a table with the following data:

ID	Name	CFU	Action
1	Information Systems	6	Add Exam Date
2	Data Mining	9	Add Exam Date

Figure 6: Example of Add Exam

- *ID*, the id of the course;
- *Name*, the name of the course;
- *CFU*, the number of credits assigned to the course;
- *Add Exam Date*, button the professor has to push in order to add an exam corresponding to the course.

If the Professor pushes one of the *Add Exam Date* buttons a confirm dialog is presented and it asks for the date of the exam to insert, fig. ???. If the Professor wants to confirm he:

1. Selects the date using the datepicker;
2. Pushes the *Confirm* button in the dialog.

To go back and undo the operation the Professor just pushes the *Delete* button.

**Date Dialog**

Insert Date

Date:

Figure 7: Example of Add Exam dialog

**Task0**

ID Utente:

Role:

What do you want to do?:

Student	Course	Date	Grade	Action
1	Information S...	2019-09-10	18	<input type="button" value="Insert Mark"/>
1	Information S...	2019-09-10	18	<input type="button" value="Insert Mark"/>
1	Information S...	2019-09-10	18	<input type="button" value="Insert Mark"/>
1	Data Mining	2019-10-10	18	<input type="button" value="Insert Mark"/>
1	Data Mining	2019-10-10	18	<input type="button" value="Insert Mark"/>
2	Information S...	2019-09-10	30	<input type="button" value="Insert Mark"/>
2	Information S...	2019-09-10	30	<input type="button" value="Insert Mark"/>
2	Information S...	2019-09-10	30	<input type="button" value="Insert Mark"/>
2	Data Mining	2019-10-10	30	<input type="button" value="Insert Mark"/>
2	Data Mining	2019-10-10	30	<input type="button" value="Insert Mark"/>

Figure 8: Example of Add Grade

## Add Grade

If the Professor select the *Add Grade* option, the table, fig. 8, is populated with a list of all the registrations to the courses he is currently holding. Each element of the list is composed of:

- *Student*, the id of the student enrolled to the exam;
- *Course*, the name of the course;
- *Date*, the date of the exam;
- *Insert Mark* button that the professor has to push in order to insert a grade to the corresponding registration.

If the Professor pushes one of the *Insert mark* buttons a confirm dialog is presented and it asks for the grade of the exam to insert, fig. 9. If the Professor wants to confirm he:

1. Inserts the grade in the corresponding field;
2. Pushes the *Confirm* button in the dialog.

To go back and undo the operation the Professor just pushes the *Delete* button.

## Student

A Student:

1. Inserts his ID number in the *UserID* field;

Mark Dialog

Insert Mark

Mark: 18

Annulla Confirm

Figure 9: Example of Add Grade dialog

2. Selects *Student* in the *Role* choice box;
3. Selects the action he wants to perform;
4. Pushes the *Confirm* button.

### Register to Exam

If the Student select the *Register to Exam* option, the table, fig. 10, is populated with a list of all the available exams. Each element of the list is composed of:

Task0

ID Utente: 1

Role: Student

What do you want to do?: Register to Exam

CONFIRM

Course	Date	Action
Process-Driven Informatio...	2019-10-31	Register

Figure 10: Example of Register to Exam

- *Course*, the name of the course;
- *Date*, the date of the exam;
- *Register*, button the student has to push in order to register to the selected exam.

If the Student pushes one of the *Register to Exam* the table is updated so that it shows all the exams to which the Student is not enrolled.

### Deregister to Exam

If the Student select the *Deregister* option, the table, fig. 11, is populated with a list of all the registrations corresponding to the Student. Each element of the list is composed of:

**Task0**

ID Utente:

Role:

What do you want to do?:

Course	Date	Grade	Action
Process-Driven Info...	2019-11-08	0	<input type="button" value="Deregister"/>
Process-Driven Info...	2019-11-08	0	<input type="button" value="Deregister"/>
Process-Driven Info...	2019-11-08	0	<input type="button" value="Deregister"/>

Figure 11: Example of Deregister to Exam

- *Course*, the name of the course of the exam the Student is enrolled to;
- *Date*, the date of the exam;
- *Deregister* button that the Student has to push in order to do the deregistration.

If the Professor pushes one of the *Deregister* the table is updated so that it shows all the exams to which the Student is not enrolled.

### See Grades

If the Student select the *See Grades* option, the table, fig. 12, is populated with a list of all the exams the student has done. Each element of the list is composed of:

- *Course*, the name of the course;
- *Date*, the date when the student passed the exam;
- *Grade*, the grade the Student got.

Task0

ID Utente

2

CONFIRM

Role

Student

What do you want to do?

See Grades

Course	Date	Grade
Information Systems	2019-09-10	30
Information Systems	2019-09-10	30
Information Systems	2019-09-10	30
Data Mining	2019-10-10	30
Data Mining	2019-10-10	30

Figure 12: Example of See Grades