Università di Federico II

Basi di dati

# Progettazione e sviluppo di una base di dati relazionale per un applicazione di e-learning

*Authors*
Luigi Penza
Stoycho Mutafchiev

*Professors*
Prof. Silvio Barra
Prof. Porfirio Tramontana

Dicembre 2021

# Indice


**1 Introduzione** **3**
- 1.1 Descrizione . . . 3
- 1.2 Analisi dei requisiti . . . 3

**2 Progettazione Concettuale** **5**
- 2.1 Introduzione . . . 5
- 2.2 Class diagram . . . 5

**3 Restrutturazione del class diagram** **6**
- 3.1 Analisi delle ridondanze . . . 6
- 3.2 Eliminazione delle generalizzazioni . . . 6
- 3.3 Eliminazione degli attributi multivalore . . . 7
- 3.4 Eliminazione degli attributi composti . . . 7
- 3.5 Partizionamento e accorpamento delle entità e associazioni . . . 7
- 3.6 Scelta degli identificatori primari . . . 7
- 3.7 Class Diagram Restrutturato . . . 7
- 3.8 Dizionario delle classi . . . 8
- 3.9 Dizionario delle associazioni . . . 10
- 3.10 Dizionario dei vincoli . . . 11

**4 Progettazione Logica** **13**
- 4.1 Schema logico . . . 13

**5 Progettazione Fisica** **15**
- 5.1 Rinominazione delle Tabelle . . . 15
- 5.2 Gestione delle Assertion . . . 15
- 5.3 Gestione delle eccezioni . . . 16
- 5.4 Automazioni . . . 16
- 5.5 Funzioni e Procedure . . . 17
- 5.6 Creazione dei domini . . . 18
- 5.7 Definizione delle tabelle . . . 19
  - 5.7.1 Professor . . . 19
  - 5.7.2 Student . . . 19
  - 5.7.3 Test . . . 20

5.7.4 Class_T . . . . . 20
5.7.5 Lecture . . . . . 21
5.7.6 Open_Quiz . . . . . 21
5.7.7 Closed_Quiz . . . . . 22
5.7.8 Take . . . . . 22
5.7.9 Test_Taken . . . . . 23
5.7.10 Open_Answer . . . . . 23
5.7.11 Closed_Answer . . . . . 24
5.8 Funzioni e Trigger . . . . . 25
5.8.1 Update_CQ_Score . . . . . 25
5.8.2 Valid_RightAnswer . . . . . 26
5.8.3 Valid_GivenAnswer . . . . . 27
5.8.4 Evaluate_Total_Score . . . . . 27
5.8.5 Test_Is_Passed . . . . . 28
5.8.6 Valid_Open_Score . . . . . 29
5.8.7 Unique_Username e Unique_Email . . . . . 30
5.8.8 Username_for_student . . . . . 31
5.8.9 Email_for_student . . . . . 31
5.8.10 Username_for_professor . . . . . 32
5.8.11 Email_for_professor . . . . . 33
5.8.12 Revise_function . . . . . 33
5.8.13 getStudent . . . . . 34
5.8.14 getProfessor . . . . . 34
5.8.15 getTest . . . . . 34
5.8.16 getClass . . . . . 34
5.8.17 getLecture . . . . . 35
5.9 Popolazione . . . . . 35

# Capitolo 1

# Introduzione

## 1.1 Descrizione

Si è deciso di costruire un applicazione di e-learning concentrandosi su un mini-word universitario. Dove gli **utenti** saranno **professori** e **studenti**. Ogni studente potrà seguire dei corsi, per poter visualizzare il materiale didattico pubblicato da un professore.

Ogni professore può pubblicare un **test** che potrà contenere **quiz** a risposta multipla o aperta. Successivamente uno studente potrà **partecipare** ad un test, ed una volta completato il professore potrà correggere i quiz a risposta aperta ed assegnare un punteggio.

## 1.2 Analisi dei requisiti

Abbiamo identificato i seguenti requisiti richiesti per la base di dati:

- Un'entità Utente che si specializzerà in altre due entità: Studente e Professore, ognuno dei quali si registrerà al sistema con nome, cognome, email, username e password;
- Un'entità Test, il cui nome è unico, per memorizzare i dati relativi ai test creati dai Professori;
- Un'entità Quiz che si specializzerà in altre due entità: Quiz a Risposta Aperta, e Quiz a Risposta Chiusa. Ogni Quiz è caratterizzato da un testo che contiene la domanda posta;
- Un Quiz a Risposta Aperta, inoltre, sarà caratterizzato da una lunghezza massima della risposta, un punteggio massimo e un punteggio minimo;

- A sua volta, un Quiz a Risposta Chiusa sarà caratterizzato da un elenco di massimo 4 risposte possibili (tra cui solo una sarà considerata quella corretta), un punteggio in caso di risposta giusta, e un punteggio in caso di risposta sbagliata;
- Un'entità che rappresenta lo svolgimento di un test da parte di uno studente;
- Un'entità che rappresenta la risposta ad un quiz a risposta aperta all'interno dello svolgimento di un test;
- Un'entità che rappresenta la risposta ad un quiz a risposta chiusa all'interno dello svolgimento di un test;
- Il sistema dovrà calcolare automaticamente il punteggio delle domande a risposta chiusa;

# Capitolo 2

# Progettazione Concettuale

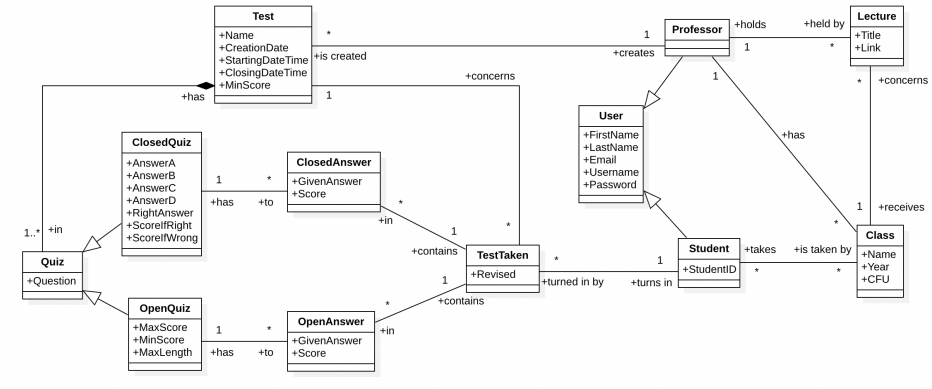
## 2.1 Introduzione

É stato scelto di implementare i **Quiz a risposta multipla**(ClosedQuiz) ed i **Quiz a risposta aperta**(OpenQuiz) come specializzazioni di una classe **Quiz**. Analogamente per **Studente** e **Professore** che sono specializzazioni di una classe **Utente**.

Un **Test** sarà quindi composto di varie domande (Quiz) e l'entità **TestTaken** ci permetterà di collegare ogni **studente** ad uno specifico **Test** ed alle sue risposte per quel **Test**.

Le entità **Class** e **Lecture** rappresentano rispettivamente il corso e le lezioni di un corso, dove è presente opzionalmente un link a del materiale extra dato dal docente.

## 2.2 Class diagram




Link per la visione del class diagram

# Capitolo 3

# Restrutturazione del class diagram

## 3.1 Analisi delle ridondanze

Nell'entità **TestTaken** è presente l'attributo **TotalScore**, che è un attributo che è possibile calcolare anche tramite la somma di tutti gli **Score** di ogni risposta dello studente. Ma è stato deciso di mantenere questo attributo secondo il seguente ragionamento, considerando un miniworld abbastanza ridotto con (tavola dei volumi):

- 20 classi
- 100 studenti
- Una media di 10 domande per test

Otteniamo che in un mese in cui vengono corretti i test il database dovrà eseguire la query del calcolo del punteggio totale, almeno $20 \times 100 \times 10 = 20.000$ volte. Cioè circa 660 volte al giorno.

È stato fatto un ragionamento analogo anche per l'attributo Passed nella stessa entità.

## 3.2 Eliminazione delle generalizzazioni

La generalizzazione di **ClosedQuiz** e **OpenQuiz**, è stata eliminata scegliendo di accorpare l'entità padre nelle due entità figlie. Il motivo è perchè questa generalizzazione era **totale e disgiunta**.

La generalizzazione di **Professor** e **Student** è stata eliminata anche in questo caso accorpando l'entità padre nelle figlie. Il motivo è lo stesso del caso precedente, ed inoltre anche per mantenere un significato concettuale (è il professore che corregge i quiz, non un utente generico o uno studente).

## 3.3 Eliminazione degli attributi multivalore

Non sono presenti attributi multivalore.

## 3.4 Eliminazione degli attributi composti

Non sono presenti attributi composti.

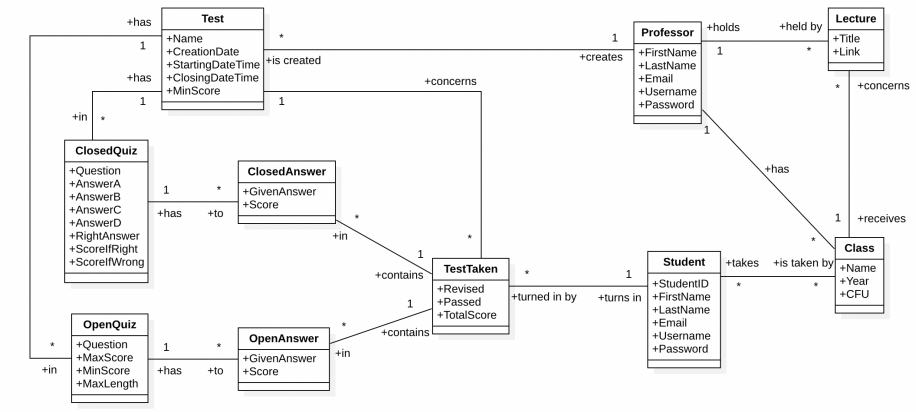
## 3.5 Partizionamento e accorpamento delle entità e associazioni

Non sono presenti associazioni da accorpare, o partizionare.

## 3.6 Scelta degli identificatori primari

Ogni **Student** è identificato dalla sua **StudentID** (matricola).
Tutte le altre entità verrano identificate tramite un codice.

## 3.7 Class Diagram Restrutturato




Link per la visione del class diagram restrutturato

## 3.8 Dizionario delle classi

Viene presentato qui il dizionario delle classi, che contiene il nome di ogni classe, i rispettivi attributi, ed il tipo, con una breve descrizione.

| Dizionario delle classi | |
|---|---|
| **NOME CLASSE** | **DESCRIZIONE** |
| **Student** | **StudentID** (*String*) : La matricola dello studente<br>**FirstName** (*String*) : Il nome dello studente<br>**LastName** (*String*) : Il cognome dello studente<br>**Email** (*String*) : L'email con cui si registra dello studente<br>**Username** (*String*) : Lo username con cui si registra lo studente<br>**Password** (*String*) : La password con cui si registra lo studente |
| **Professor** | **FirstName** (*String*) : Il nome del professore<br>**LastName** (*String*) : Il cognome del professore<br>**Email** (*String*) : L'email con cui si registra il professore<br>**Username** (*String*) : Lo username con cui si registra il professore<br>**Password** (*String*) : La password con cui si registra il professore |
| **Lecture** | **Title** (*String*) : Il titolo della lezione<br>**Link** (*String*) : Un link ad un sito esterno dove è presente del materiale didattico rigardante la lezione |
| **Class** | **Name** (*String*) : Il nome del corso<br>**Year** (*Date*) : L'anno accademico in cui il corso è svolto<br>**CFU** (*Int*) : Il numero di Crediti Formativi Universitari dedicati al corso |
| **Test** | **Name** (*String*) : Il nome del test<br>**CreationDatetime** (*Date*) : La data in cui il test viene creato<br>**DtartingDatetime** (*Date*) : La data e l'ora di inizio del test<br>**ClosingDatetime** (*Date*) : La data e l'ora di fine del test<br>**MinScore** (*Float*) : Il punteggio minimo per passare il test |

| Continuo del dizionario delle classi | |
|---|---|
| **NOME CLASSE** | **ATTRIBUTI** |
| **Open Quiz** | **Question** (*String*) : Il testo della domanda a risposta aperta<br>**MaxScore** (*Float*) : Il massimo punteggio che si può ottenere<br>rispondendo alla domanda<br>**MinScore** (*Float*) : Il minimo punteggio che si può ottenere rispondendo correttamente alla domanda<br>**MaxLength** (*Int*) : La massima lunghezza della risposta |
| **Closed Quiz** | **Question** (*String*) : Il testo della domanda a risposta chiusa<br>**AnswerA** (*String*) : Una possibile risposta, questo campo è totale<br>**AnswerB** (*String*) : Una possibile risposta, questo campo è totale<br>**AnswerC** (*String*) : Una possibile risposta, questo campo è parziale<br>**AnswerD** (*String*) : Una possibile risposta, questo campo è parziale<br>**RightAnswer** (*Char*) : La risposta corretta tra le quattro possibili<br>**ScoreIfRight** (*Float*) : Il punteggio che si ottiene rispondendo correttamente<br>**ScoreIfWrong** (*Float*) : Il punteggio che si ottiene non rispondendo correttamente |
| **Closed Answer** | **Score** (*Float*) : Il punteggio che lo studente ha ottenuto<br>**GivenAnswer** (*Char*) : La risposta, tra le possibili, data dallo studente |
| **Open Answer** | **Score** (*Float*) : La risposta data dallo studente<br>**GivenAnswer** (*String*) : Il punteggio che lo studente ha ottenuto |
| **Test Taken** | **Revised** (*Boolean*) : Un valore che indica se il test è stato corretto dal professore oppure no<br>**Passed** (*Boolean*) : Indica se lo studente ha passato quel test<br>**TotalScore** (*Float*) : Il punteggio ottenuto dallo studente in quel test |

## 3.9 Dizionario delle associazioni

Viene qui presentato il dizionario delle associazioni, che contiene tutte le associazioni, i rispettivi ruoli, ed una breve descrizione dell'associazione.

| Dizionario delle associazioni | |
|---|---|
| **NOME ASSOCIAZIONE** | **DESCRIZIONE** |
| **Test Creation** | **Test** [*] *ruolo* **is created** : Rappresenta il test che viene creato.<br>**Professor** [1] *ruolo* **creates** : Rappresenta il professore che crea il test. |
| **Lecture Holding** | **Professor** [1] *ruolo* **holds** : Rappresenta il professore che tiene la lezione.<br>**Lecture** [*] *ruolo* **held by** : Rappresenta la lezione che tenuta fatta dal prof. |
| **Lecture To** | **Lecture** [*] *ruolo* **concerns** : Rappresenta la lezione per il corso.<br>**Class** [1] *ruolo* **receives** : Rappresenta il corso a cui è destinata la lezione. |
| **Attending** | **Class** [*] *ruolo* **is taken by** : Rappresenta il corso frequentato dallo studente.<br>**Student** [*] *ruolo* **takes** : Rappresenta lo studente che frequenta il corso. |
| **Teaching** | **Class** [*] *ruolo* **has** : Rappresenta il corso tenuto dal professore.<br>**Professor** [1] *ruolo* **has** : Rappresenta il professore che tiene il corso. |
| **Turns In** | **Student** [1] *ruolo* **turns in** : Rappresenta lo studente che consegna il suo test svolto.<br>**TestTaken** [*] *ruolo* **turned in by** : Rappresenta il test svolto e consegnato dallo studente. |
| **Takes Of Test** | **TestTaken** [*] *ruolo* **concerns** : Rappresenta il test svolto collegato ad un determinato test<br>**Test** [1] *ruolo* **concerns** : Rappresenta il test a cui è collegato il test svolto |
| **Contains Open Answer** | **TestTaken** [1] *ruolo* **contains** : Rappresenta il test svolto che contiene risposte aperte.<br>**OpenAnswer** [*] *ruolo* **in** : Rappresenta la risposta aperta contenuta in un test svolto. |
| **Contains Closed Answer** | **TestTaken** [1] *ruolo* **contains** : Rappresenta il test svolto che contiene risposte chiuse.<br>**ClosedAnswer** [*] *ruolo* **in** : Rappresenta la risposta chiusa contenuta in un test svolto. |

| Continuo del dizionario delle associazioni | |
|---|---|
| **NOME ASSOCIAZIONE** | **DESCRIZIONE** |
| **Open Answer To** | **OpenAnswer** [*] *ruolo* **to** : Rappresenta la risposta al quiz a risposta aperta.<br>**OpenQuiz** [1] *ruolo* **has** : Rappresenta il quiz a risposta aperta a cui si riferisce la risposta. |
| **Closed Answer To** | **ClosedAnswer** [*] *ruolo* **to** : Rappresenta la risposta al quiz a risposta chiusa.<br>**ClosedQuiz** [1] *ruolo* **has** : Rappresenta il quiz a risposta chiusa a cui si riferisce la risposta. |
| **Has Closed Quiz** | **ClosedQuiz** [*] *ruolo* **in** : Rappresenta il quiz a risposta chiusa contenuto in un test.<br>**Test** [1] *ruolo* **has** : Rappresenta il test che contiene i quiz a risposta chiusa. |
| **Has Open Quiz** | **OpenQuiz** [*] *ruolo* **in** : Rappresenta il quiz a risposta aperta contenuto in un test.<br>**Test** [1] *ruolo* **has** : Rappresenta il test che contiene i quiz a risposta aperta. |

## 3.10 Dizionario dei vincoli

Vengono qui proposti i vincoli che saranno presenti nel database, con la rispettiva descrizione.

| Dizionario dei vincoli | |
|---|---|
| **NOME VINCOLO** | **DESCRIZIONE** |
| **Unique Username** | Non devono esistere più utenti con lo stesso username. |
| **Unique Email** | Non devono esistere più utenti con la stessa email. |
| **Unique NameTest** | Non devono esistere più test con lo stesso nome. |
| **Valid Name** | I nomi non devono contenere numeri. Inoltre devono avere almeno 1 carattere e al più 35 caratteri. |
| **Valid Email** | La mail deve avere la forma di u@v.w con u, v, w stringhe non nulle. |
| **Valid StartingDateTime** | La data di inizio del test deve essere successiva al giorno in cui viene creato il test. |
| **Valid ClosingDateTime** | La differenza tra ClosingDateTime e StartingDateTime deve essere maggiore o uguale di 10 minuti. |
| **Valid MaxLength** | La lunghezza massima deve essere maggiore di 0. |
| **Valid Right Answer** | La risposta di una domanda multipla deve tra quelle possibili (a, b, [c], [d]). |
| **Valid Question** | La domanda deve essere non nulla. |

| Continuo del dizionario dei vincoli | |
|---|---|
| **NOME VINCOLO** | **DESCRIZIONE** |
| **Strong Password** | La password deve essere composta da più di 8 caratteri, almeno una lettera, almeno un numero ed almeno carattere speciale. |
| **Valid CFU** | Il numero di CFU deve essere compreso tra 1 e 20. |
| **Valid Given Answer** | La risposta di una domanda aperta non deve avere più caratteri di quanti concessi per quella domanda. |
| **Valid Open Score** | Il Il punteggio dato alla risposta aperta, deve essere compreso tra maxScore e minScore. |

# Capitolo 4

# Progettazione Logica

## 4.1 Schema logico

In questo schema relazionale le chiavi primarie sono state scritte in **grassetto**, mentre le chiavi esterne vengono scritte in *corsivo*.

| | |
|---|---|
| **TEST** | (**CodTest**, Name, CreationDateTime, StartingDateTime, ClosingDateTime, MinScore, *CodP*)<br>*CodP* $\mapsto$ PROFESSOR.CodP |
| **PROFESSOR** | (**CodP**, FirstName, LastName, Email, Username, Password) |
| **LECTURE** | (**CodL** ,Title, Link, *CodP*, *CodC*)<br>*CodP* $\mapsto$ PROFESSOR.CodP<br>*CodC* $\mapsto$ CLASS.CodC |
| **OPENQUIZ** | (**CodOQ**, Question, MaxScore, MinScore, MaxLength, *CodTest*)<br>*CodTest* $\mapsto$ TEST.CodTest |
| **CLOSEDQUIZ** | (**CodCQ**, Question, AnswerA, AnswerB, AnswerC, AnswerD, RightAnswer, ScoreIfRight, ScoreIfWrong, *CodTest*)<br>*CodTest* $\mapsto$ TEST.CodTest |
| **CLASS** | (**CodC**, Name, Year, CFU, *CodP*)<br>*CodP* $\mapsto$ PROFESSOR.CodP |
| **TAKE** | (***CodC***, ***StudentID***)<br>*CodC* $\mapsto$ CLASS.CodC<br>*StudentID* $\mapsto$ STUDENT.StudentID |

| | |
|---|---|
| **OPENANSWER** | (**CodOA**, GivenAnswer, Score, *CodOQ*, *CodTestTaken*)<br>*CodOQ* $\mapsto$ OPENQUIZ.CodOQ<br>*CodTestTaken* $\mapsto$ TESTTAKEN.CodTestTaken |
| **CLOSEDANSWER** | (**CodCA**, GivenAnswer, Score, *CodCQ*, *CodTestTaken*)<br>*CodCQ* $\mapsto$ CLOSEDQUIZ.CodCQ<br>*CodTestTaken* $\mapsto$ TESTTAKEN.CodTestTaken |
| **STUDENT** | (**StudentID**, FirstName, LastName, Email, Username, Password) |
| **TESTTAKEN** | (**CodTestTaken**, Revised, Passed, TotalScore, *CodTest*, *StudentID*)<br>*CodTest* $\mapsto$ TEST.CodTest<br>*StudentID* $\mapsto$ STUDENT.StudentID |

# Capitolo 5

# Progettazione Fisica

## 5.1 Rinominazione delle Tabelle

Nella progettazione fisica delle classi ci sono state delle modifiche rispetto il diagramma restrutturato. Sono state **rinominate le seguenti tabelle** per una maggiore legibilità del codice:

- TESTTAKEN → **TEST_TAKEN**
- CLOSEDANSWER → **CLOSED_ANSWER**
- OPENANSWER → **OPEN_ANSWER**
- CLOSEDQUIZ → **CLOSED_QUIZ**
- OPENQUIZ → **OPEN_QUIZ**

## 5.2 Gestione delle Assertion

Dato che postgreSQL non permette l'uso di **assertion**, esse sono state implementate concettualmente tramite procedure; Ecco una lista di vincoli implementati tramite procedure:

- **Valid_GivenAnswe**r: La lunghezza della risposta data NON deve superare MaxLength dell'OpenQuiz associato;
- **Valid_Right_Answer** : La risposta di una domanda multipla deve tra quelle possibili;
- **Valid_Open_Score** : Il punteggio dato alla risposta aperta, deve essere compreso tra maxScore e minScore.
- **Unique_Username** : Non devono esistere più utenti con lo stesso username;
- **Unique_Email** : Non devono esistere più utenti con la stessa email.

## 5.3 Gestione delle eccezioni

Inoltre per una migliore gestione degli errori, sono stati creati dei **codici** per la **gestione delle eccezioni**:

- **E000C** (Errore 000 C) : Eccezione nel caso in cui l'utente abbia risposta 'c' ad una domanda a risposta multipla, quando quella non è presente tra le possibili risposte;
- **E000D** (Errore 000 D) : Eccezione nel caso in cui l'utente abbia risposta 'd' ad una domanda a risposta multipla, quando quella non è presente tra le possibili risposte;
- **T00LG** (Too long) : Eccezione nel caso in cui l'utente abbia risposto con un numero di caratteri maggiori di quelli consentiti dalla domanda;
- **SF001** (Select fault 001) : Eccezione se la select restituisce più di una tupla;
- **SNIMM** (Score not in min max) : Eccezione se il voto inserito dal professore, per una domanda a risposta aperta, non è compreso tra quelli consentiti;
- **UNALP** (UserName ALready exists in Professor): Eccezione lanciata nel caso in cui uno Studente vuole inserire un username già esistente nella base di dati, in particolare tra i Professori.
- **EMALP** (EMail ALready exists in Professor): Eccezione lanciata nel caso in cui uno Studente vuole inserire una Email già esistente nella base di dati, in particolare tra i Professori;
- **UNALS** (UserName ALready exists in Student): Eccezione lanciata se un professore vuole inserire un username che già appartiene ad uno studente.
- **EMALS** (EMail ALready exists in Student): Eccezione lanciata se un professore vuole inserire un email che già appartiene ad uno studente.

## 5.4 Automazioni

Infine il database fornisce anche le seguenti automazioni:

- **Update_Closed_Quiz_Score** : Correzione automatica delle domande a risposta chiusa;
- **Evaluate_Total_Score** : Quando viene corretta una risposta, viene aggiornato il risultato totale;
- **Test_Is_Passed** : Quando il total score supera il min score, allora lo studente ha passato il test;

## 5.5 Funzioni e Procedure

Queste procedure potranno essere usate tramite l'applicativo per ottenere informazioni.

- **revise_function** : Inserito un test, ed un professore, "committa" tutti i test_taken, di quel test (ovviamente se il test è stato creato da quel professore);
- **username_for_student** : Inserito un nuovo Username ed uno StudentID, cambia l'username se nessun altro utente sta usando quell'username;
- **email_for_student** : Inserita un nuova Email ed uno StudentID, cambia la mail se nessun altro utente sta usando quella mail;
- **username_for_professor** : Inserito un nuovo Username ed un CodP, cambia l'username se nessun altro utente sta usando quell'username;
- **email_for_professor** : Inserita un nuova Email ed un CodP, cambia la mail se nessun altro utente sta usando quella mail;
- **getStudent** : Dato in input lo Username di uno studente restituisce il rispettivo StudentID;
- **getProfessor** : Dato in input lo Username di un professore restituisce il rispettivo CodP;
- **getTest** : Dato in input il nome di un Test restituisce il rispettivo CodTest;
- **getLecture** : Dato il titolo di una lezione, restituisce il codice della stessa;

Consigliamo la lettura del codice dal seguente link : Github.

## 5.6 Creazione dei domini

E' stato scelto di implementare dei domini per due motivi: per una più semplice lettura delle tabelle, e per scalabilità (se si volesse per esempio cambiare la lunghezza di un nome, basterebbe cambiare il dominio **PERSON_NAME**). Ecco i domini implementati:

```
1  -- Valid_Name : I nomi non devono contenere numeri e devono avere
       almeno 1 carattere e al piu' 35 caratteri.
2  CREATE DOMAIN PERSON_NAME AS VARCHAR(35)
3    CHECK ( VALUE <> '' AND VALUE NOT SIMILAR TO '%[0-9]+%' );
4
5  -- Valid_Test_Name : Il nome del test pu  avere un numero di
       caratteri compreso tra 1 e 55.
6  CREATE DOMAIN TEST_NAME AS VARCHAR(55)
7    CHECK ( VALUE <> '' );
8
9  -- Valid_Email : La mail deve avere la forma di u@v.w con u, v, w
       stringhe non nulle
10 CREATE DOMAIN EMAIL AS VARCHAR(254)
11   CHECK ( VALUE LIKE '_%@_%._%' );
12
13 -- Strong_Password : La password deve essere composta da pi  di 8
       caratteri, almeno una lettera, almeno un numero ed almeno
       carattere speciale (!" $ %&/()=_:;,.-+*#)
14 CREATE DOMAIN PASSWORD_D AS VARCHAR(128)
15   CHECK (VALUE ~ '^.*(?=.*[a-zA-Z])(?=.*[0-9])(?=.*[!@#$%^&*]).*$'
16     AND VALUE LIKE '________%');
17
18 -- Valid_Right_Answer : La risposta di una domanda multipla deve
       tra quelle possibili (Dominio = {'a', 'b', 'c', 'd'})
19 CREATE DOMAIN CLOSED_ANSWER_D AS CHAR(1)
20   CHECK ( VALUE IN ('a', 'b', 'c', 'd') ); -- Questo vincolo verr
        implementato ulteriormente tramite funzione
21
22 -- Valid_CFU : Il numero di CFU deve essere compreso tra 1 e 20
23 CREATE DOMAIN VALID_CFU AS INTEGER
24   CHECK ( VALUE BETWEEN 1 AND 20 );
25
26 -- Dominio per i punteggi dei quiz
27 CREATE DOMAIN SCORE_D AS FLOAT ;
```

## 5.7 Definizione delle tabelle

### 5.7.1 Professor

```
-- Tabella PROFESSOR

CREATE TABLE PROFESSOR(
  CodP SERIAL NOT NULL,
  FirstName PERSON_NAME NOT NULL,
  LastName PERSON_NAME NOT NULL,
  Email EMAIL UNIQUE NOT NULL,
  Username VARCHAR(35) UNIQUE NOT NULL,
  Pw PASSWORD_D NOT NULL
);
-- Aggiunta del vincolo di chiave primaria
ALTER TABLE PROFESSOR
  ADD CONSTRAINT professor_pk PRIMARY KEY(CodP);
```

### 5.7.2 Student

```
-- Tabella STUDENT

CREATE TABLE STUDENT(
  StudentID SERIAL NOT NULL,
  FirstName PERSON_NAME NOT NULL,
  LastName PERSON_NAME NOT NULL,
  Email EMAIL UNIQUE NOT NULL,
  Username VARCHAR(35) UNIQUE NOT NULL,
  Pw PASSWORD_D NOT NULL
);
-- Aggiunta del vincolo di chiave primaria
ALTER TABLE STUDENT
  ADD CONSTRAINT student_pk PRIMARY KEY(StudentID);
```

### 5.7.3 Test

```
-- Tabella TEST

CREATE TABLE TEST(
  CodTest SERIAL NOT NULL,
  Name TEST_NAME UNIQUE NOT NULL,
  CreationDateTime TIMESTAMP DEFAULT LOCALTIMESTAMP,
  StartingDateTime TIMESTAMP,
  ClosingDateTime TIMESTAMP,
  MinScore SCORE_D,
  CodP SERIAL NOT NULL
);
-- Aggiunta del vincolo di chiave primaria
ALTER TABLE TEST
  ADD CONSTRAINT test_pk PRIMARY KEY(CodTest),
  -- Aggiunta del vincolo di chiave esterna sulla tabella PROFESSOR
  ADD CONSTRAINT test_fk FOREIGN KEY(CodP) REFERENCES PROFESSOR(
    CodP)
    -- Quando il codice del professore cambia, viene cambiato anche
     in TEST
    ON UPDATE CASCADE
    ON DELETE RESTRICT; --

-- Valid_Starting_Date_Time : La data di inizio del test deve
    essere successiva al giorno in cui viene creato il test
ALTER TABLE TEST
ADD CONSTRAINT Valid_Starting_DateTime
  CHECK ( StartingDateTime > CreationDateTime );

-- Valid_ClosingDateTime : La differenza tra ClosingDateTime e
    StartingDateTime deve essere maggiore o uguale di 10 minuti
ALTER TABLE TEST
ADD CONSTRAINT Valid_ClosingDateTime
  CHECK ( DATE_PART('minute', ClosingDateTime - StartingDateTime )
    >= 10);
```

### 5.7.4 Class_T

```
-- TABELLA CLASS_T

CREATE TABLE CLASS_T(
  CodC SERIAL NOT NULL,
  Name VARCHAR(50) NOT NULL UNIQUE,
  Year INT DEFAULT DATE_PART('year', LOCALTIMESTAMP),
  CFU VALID_CFU NOT NULL,
  CodP SERIAL
);
-- Aggiunta del vincolo di chiave primaria
ALTER TABLE CLASS_T
  ADD CONSTRAINT class_pk PRIMARY KEY(CodC),
  -- Aggiunta del vincolo di chiave esterna sulla tabella PROFESSOR
  ADD CONSTRAINT class_fk FOREIGN KEY(CodP) REFERENCES PROFESSOR(
    CodP)
    ON UPDATE CASCADE
    ON DELETE RESTRICT;
```

### 5.7.5 Lecture

```
-- TABELLA LECTURE

CREATE TABLE LECTURE(
  CodL SERIAL NOT NULL,
  Title VARCHAR(30) NOT NULL UNIQUE,
  Link VARCHAR(512),
  CodP SERIAL NOT NULL,
  CodC SERIAL NOT NULL
);
-- Aggiunta del vincolo di chiave primaria
ALTER TABLE LECTURE
  ADD CONSTRAINT lecture_pk PRIMARY KEY(CodL),
  -- Aggiunta del vincolo di chiave esterna sulla tabella PROFESSOR
  ADD CONSTRAINT lecture_professor_fk
    FOREIGN KEY(CodP) REFERENCES PROFESSOR(CodP)
      ON UPDATE CASCADE
      ON DELETE RESTRICT,
  -- Aggiunta del vincolo di chiave esterna sulla tabella CLASS_T
  ADD CONSTRAINT lecture_class_fk FOREIGN KEY(CodC) REFERENCES
    CLASS_T(CodC)
    ON UPDATE CASCADE
    ON DELETE RESTRICT;
```

### 5.7.6 Open_Quiz

```
-- TABELLA OPEN_QUIZ

CREATE TABLE OPEN_QUIZ(
  CodOQ SERIAL NOT NULL,
  Question VARCHAR(512) NOT NULL,
  MaxScore SCORE_D NOT NULL,
  MinScore SCORE_D NOT NULL,
  MaxLength INT DEFAULT 1024,
  CodTest SERIAL NOT NULL
);
-- Aggiunta del vincolo di chiave primaria
ALTER TABLE OPEN_QUIZ
  ADD CONSTRAINT open_quiz_pk PRIMARY KEY(CodOQ),
  -- Aggiunta del vincolo di chiave esterna sulla tabella TEST
  ADD CONSTRAINT open_quiz_fk FOREIGN KEY(CodTest) REFERENCES TEST(
    CodTest)
    ON UPDATE CASCADE
    ON DELETE RESTRICT;

-- MaxLength_UpperBound : La possibilita' della lughezza della
    risposta aperta deve essere compresa tra 1 e 1024
ALTER TABLE OPEN_QUIZ
  ADD CONSTRAINT MaxLength_UpperBound
    CHECK ( MaxLength BETWEEN 1 AND 1024 );
```

### 5.7.7 Closed_Quiz

```
-- TABELLA CLOSED_QUIZ

CREATE TABLE CLOSED_QUIZ(
  CodCQ SERIAL NOT NULL,
  Question VARCHAR(512) NOT NULL,
  AnswerA VARCHAR(128) NOT NULL,
  AnswerB VARCHAR(128) NOT NULL,
  AnswerC VARCHAR(128),
  AnswerD VARCHAR(128),
  RightAnswer CLOSED_ANSWER_D NOT NULL,
  ScoreIfRight SCORE_D NOT NULL,
  ScoreIfWrong SCORE_D NOT NULL,
  CodTest SERIAL NOT NULL
);
-- Aggiunta della chiave primaria, e della chiave esterna sulla
    tabella TEST
ALTER TABLE CLOSED_QUIZ
  ADD CONSTRAINT closed_quiz_pk PRIMARY KEY(CodCQ),
  ADD CONSTRAINT closed_quiz_fk FOREIGN KEY(CodTest) REFERENCES
    TEST(CodTest)
    ON UPDATE CASCADE
    ON DELETE RESTRICT;
```

### 5.7.8 Take

```
-- TABELLA TAKE

CREATE TABLE TAKE(
  CodC SERIAL NOT NULL,
  StudentID SERIAL NOT NULL
);
-- Aggiunta della chiave primaria
ALTER TABLE TAKE
  ADD CONSTRAINT take_pk PRIMARY KEY(CodC, StudentID),
  -- Aggiunta del vincolo di chiave esterna sulla tabella CLASS_T
  ADD CONSTRAINT take_class_fk FOREIGN KEY(CodC) REFERENCES CLASS_T
    (CodC)
    ON UPDATE CASCADE
    ON DELETE RESTRICT,
  -- Aggiunta del vincolo di chiave esterna sulla tabella STUDENT
  ADD CONSTRAINT take_student_fk
    FOREIGN KEY(StudentID) REFERENCES STUDENT(StudentID)
      ON UPDATE CASCADE
      ON DELETE RESTRICT;
```

### 5.7.9 Test_Taken

```
-- TABELLA TEST_TAKEN

CREATE TABLE TEST_TAKEN(
  CodTestTaken SERIAL NOT NULL,
  CodTest SERIAL NOT NULL,
  StudentID SERIAL NOT NULL,
  Revised BOOLEAN DEFAULT FALSE,
  Passed BOOLEAN,
  TotalScore SCORE_D DEFAULT 0
);
-- Aggiunta della chiave primaria, e delle chiavi esterne su TEST e
    STUDENT
ALTER TABLE TEST_TAKEN
  ADD CONSTRAINT test_taken_pk PRIMARY KEY(CodTestTaken),
  -- Ogni studente non pu  consegnare pi  volte lo stesso test
  ADD CONSTRAINT unique_student_test UNIQUE(StudentID, CodTest),
  -- Aggiunta del vincolo di chiave esterna sulla tabella TEST
  ADD CONSTRAINT test_taken_test_fk
    FOREIGN KEY(CodTest) REFERENCES TEST(CodTest)
      ON UPDATE CASCADE
      ON DELETE RESTRICT,
  -- Aggiunta del vincolo di chiave esterna sulla tabella STUDENT
  ADD CONSTRAINT test_taken_student
    FOREIGN KEY(StudentID) REFERENCES STUDENT(StudentID)
      ON UPDATE CASCADE
      ON DELETE RESTRICT;
```

### 5.7.10 Open_Answer

```
-- TABELLA OPEN_ANSWER

CREATE TABLE OPEN_ANSWER(
  CodOA SERIAL NOT NULL,
  GivenAnswer VARCHAR(1024),
  Score SCORE_D DEFAULT 0,
  CodOQ SERIAL NOT NULL,
  CodTest_Taken SERIAL NOT NULL
);
-- Aggiunta della chiave primaria
ALTER TABLE OPEN_ANSWER
  ADD CONSTRAINT open_answer_pk PRIMARY KEY(CodOA),
  -- Aggiunta del vincolo di chiave esterna sulla tabella OPEN_QUIZ
  ADD CONSTRAINT open_answer_open_quiz_fk
    FOREIGN KEY(CodOQ) REFERENCES OPEN_QUIZ(CodOQ)
      ON UPDATE CASCADE
      ON DELETE RESTRICT,
  -- Aggiunta del vincolo di chiave esterna sulla tabella
    TEST_TAKEN
  ADD CONSTRAINT open_answer_test_taken_fk
    FOREIGN KEY(CodTest_Taken) REFERENCES TEST_TAKEN(CodTestTaken)
      ON UPDATE CASCADE
      ON DELETE RESTRICT;
```

### 5.7.11 Closed_Answer

```
-- TABELLA CLOSED_ANSWER

CREATE TABLE CLOSED_ANSWER(
  CodCA SERIAL NOT NULL,
  GivenAnswer CHAR,
  Score SCORE_D DEFAULT 0,
  CodCQ SERIAL NOT NULL,
  CodTest_Taken SERIAL NOT NULL
);
-- Aggiunta della chiave primaria
ALTER TABLE CLOSED_ANSWER
  ADD CONSTRAINT closed_answer_pk PRIMARY KEY(CodCA),
  -- Aggiunta del vincolo di chiave esterna sulla tabella
    CLOSED_QUIZ
  ADD CONSTRAINT closed_answer_closed_quiz_fk
    FOREIGN KEY(CodCQ) REFERENCES CLOSED_QUIZ(CodCQ)
      ON UPDATE CASCADE
      ON DELETE RESTRICT,
  -- Aggiunta del vincolo di chiave esterna sulla tabella
    TEST_TAKEN
  ADD CONSTRAINT closed_answer_test_taken_fk
    FOREIGN KEY(CodTest_Taken) REFERENCES TEST_TAKEN(CodTestTaken)
      ON UPDATE CASCADE
      ON DELETE RESTRICT;
```

## 5.8 Funzioni e Trigger

### 5.8.1 Update_CQ_Score

```
-- Update_Closed_Quiz_Score : Correzione automatica risposte chiuse
CREATE OR REPLACE FUNCTION UCQS_function() RETURNS TRIGGER AS
    $Update_CQ_Score$
DECLARE
    ScoreRight CLOSED_QUIZ.ScoreIfRight%TYPE;
    ScoreWrong CLOSED_QUIZ.ScoreIfWrong%TYPE;
    RA CLOSED_QUIZ.RightAnswer%TYPE; -- Risposta corretta
  info INT := 0; -- Flag che mi dice se posso eseguire le query
BEGIN
  -- Controllo che la query restituisca una sola tupla
  SELECT COUNT(*) INTO info
    FROM CLOSED_ANSWER CA JOIN CLOSED_QUIZ CQ ON CA.CodCQ = CQ.
    CodCQ
    WHERE CA.CodCA = NEW.CodCA;

  IF info = 1 THEN
    -- Salvo i valori
      SELECT ScoreIfRight, ScoreIfWrong, RightAnswer
      INTO ScoreRight, ScoreWrong, RA
      FROM CLOSED_ANSWER CA JOIN CLOSED_QUIZ CQ ON CA.CodCQ = CQ.
    CodCQ
      WHERE CA.CodCA = NEW.CodCA;

      IF (NEW.GivenAnswer = RA) THEN -- Aggiorno se la risposta
    corretta
           UPDATE CLOSED_ANSWER
           SET Score = ScoreRight
           WHERE CodCA = NEW.CodCA;
    END IF;

    -- Aggiorno se la risposta    sbagliata
      IF (NEW.GivenAnswer <> RA AND NEW.GivenAnswer IS NOT NULL)
    THEN
           UPDATE CLOSED_ANSWER
           SET Score = ScoreWrong
           WHERE CodCA = NEW.CodCA;
      END IF;
  END IF;
  RETURN NEW;

EXCEPTION
  WHEN OTHERS THEN
    RAISE NOTICE 'SQLSTATE : %', SQLSTATE;
    RETURN NULL;
END; $Update_CQ_Score$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER Update_CQ_Score
AFTER INSERT ON CLOSED_ANSWER
FOR EACH ROW
EXECUTE PROCEDURE UCQS_function();
```

### 5.8.2 Valid_RightAnswer

```
-- Valid_Right_Answer : La risposta di una domanda multipla deve essere tra quelle possibili
CREATE OR REPLACE FUNCTION VRA_Function() RETURNS TRIGGER AS $Valid_Right_Answer$
DECLARE
  -- Indicano le rispettive risposte a quella domanda
  -- (se la risposta    opzionale resteranno NULL)
  flagC CLOSED_QUIZ.AnswerC%TYPE := NULL;
  flagD CLOSED_QUIZ.AnswerD%TYPE := NULL;
BEGIN
  -- Prendo il valore della risposta C
  SELECT AnswerC INTO flagC
  FROM CLOSED_QUIZ
  WHERE CodCQ = NEW.CodCQ;
  -- Se    stata inserita la risposta C ma non era una risposta possibile
  IF NEW.GivenAnswer = 'c' AND flagC IS NULL THEN
    RAISE EXCEPTION USING ERRCODE='E000C';
  END IF;

  -- Prendo il valore della risposta D
  SELECT AnswerD INTO flagD
  FROM CLOSED_QUIZ
  WHERE CodCQ = NEW.CodCQ;
  -- Se    stata inserita la risposta D ma non era una risposta possibile
  IF NEW.GivenAnswer = 'd' AND flagD IS NULL THEN
    RAISE EXCEPTION USING ERRCODE='E000D';
  END IF;

  RETURN NEW;

EXCEPTION
    WHEN SQLSTATE 'E000C' THEN -- E000C errore per c
    DELETE FROM CLOSED_ANSWER WHERE CodCA = NEW.CodCA;
    RAISE NOTICE 'La risposta "C" non    tra quelle possibili ';
    RETURN NULL;

  WHEN SQLSTATE 'E000D' THEN -- E000D errore per d
    DELETE FROM CLOSED_ANSWER WHERE CodCA = NEW.CodCA;
    RAISE NOTICE 'La risposta "D" non    tra quelle possibili ';
    RETURN NULL;

  WHEN OTHERS THEN
    RAISE NOTICE 'SQLSTATE : %', SQLSTATE;
    RETURN NULL;
END; $Valid_Right_Answer$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER Valid_Right_Answer
AFTER INSERT ON CLOSED_ANSWER
FOR EACH ROW
EXECUTE PROCEDURE VRA_Function();
```

### 5.8.3 Valid_GivenAnswer

```
-- Valid_GivenAnswer: La lunghezza della risposta data NON deve superare MaxLength dell'OpenQuiz associato
CREATE OR REPLACE FUNCTION VGA_function() RETURNS TRIGGER AS $$
DECLARE
  len INT;
BEGIN
  -- Cerco la massima lunghezza della risposta
  SELECT MaxLength INTO len
  FROM OPEN_ANSWER AS OA, OPEN_QUIZ AS OQ
  WHERE OA.CodOQ = OQ.CodOQ AND OA.CodOA = NEW.CodOA;

  IF LENGTH(NEW.GivenAnswer) > len THEN
    RAISE EXCEPTION USING ERRCODE='TOOLG';
  END IF;

  RETURN NEW;

EXCEPTION
  WHEN SQLSTATE 'TOOLG' THEN
    DELETE FROM OPEN_ANSWER WHERE CodOA = NEW.CodOA;
    RAISE NOTICE 'ERRORE! Risposta troppo lunga!';
    RETURN NULL;
END; $$ LANGUAGE PLPGSQL;

CREATE OR REPLACE TRIGGER Valid_GivenAnswer
AFTER INSERT ON OPEN_ANSWER
FOR EACH ROW
EXECUTE PROCEDURE VGA_function();
```

### 5.8.4 Evaluate_Total_Score

```
-- Evaluate_Total_Score : Quando viene corretta una risposta, viene aggiornato il risultato totale.
CREATE OR REPLACE FUNCTION ETS_function() RETURNS TRIGGER AS $Evaluate_Total_Score$
BEGIN
  UPDATE TEST_TAKEN
  SET TotalScore = TotalScore + NEW.Score - OLD.Score
  WHERE NEW.CodTest_Taken = CodTestTaken;
  RETURN NEW;
END; $Evaluate_Total_Score$ LANGUAGE PLPGSQL;

CREATE OR REPLACE TRIGGER Evaluate_Total_Score_Open
AFTER UPDATE OF Score ON OPEN_ANSWER
-- Update_Closed_Quiz_Score si occupa dell'update dell'attributo
FOR EACH ROW
EXECUTE PROCEDURE ETS_function();

CREATE OR REPLACE TRIGGER Evaluate_Total_Score_Closed
AFTER UPDATE OF Score ON CLOSED_ANSWER
FOR EACH ROW
EXECUTE PROCEDURE ETS_function();
```

### 5.8.5 Test_Is_Passed

```
-- Test_Is_Passed : Quando il total score supera il min score, allora lo studente ha passato il test
-- Ogni volta che viene cambiato il total score, controllo se il test    stato passato
-- Uno studente pu  aver passato un test, anche se non    stato ancora corretto dal prof
-- Potrebbe bastare rispondere alle domande a risposta chiusa
CREATE OR REPLACE FUNCTION TIP_function() RETURNS TRIGGER AS $Test_Is_Passed$
DECLARE
    tts CURSOR IS
        SELECT *
        FROM TEST_TAKEN
        WHERE TotalScore IS NOT NULL;
    hatepostgre TEST.MinScore%TYPE;
BEGIN
    -- Scorro tutti i test_taken, e vedo quali hanno totalscore superiore al minscore
    FOR i IN tts LOOP
        SELECT MinScore INTO hatepostgre FROM TEST WHERE CodTest = i.CodTest; -- Prendo il rispettivo MinScore

        -- Se il punteggio totale    maggiore del minimo, o Se non c'  , il test    banalmente passato
        IF i.TotalScore >= hatepostgre OR hatepostgre IS NULL THEN
            UPDATE TEST_TAKEN SET Passed = true WHERE CodTestTaken = i.CodTestTaken;
        ELSE --i.TotalScore < hatepostgre
            UPDATE TEST_TAKEN SET Passed = false WHERE CodTestTaken = i.CodTestTaken;
        END IF;
    END LOOP;

  RETURN NULL;
END; $Test_Is_Passed$ LANGUAGE PLPGSQL;

CREATE OR REPLACE TRIGGER Test_Is_Passed AFTER UPDATE OF TotalScore ON TEST_TAKEN
EXECUTE PROCEDURE TIP_function();
```

### 5.8.6 Valid_Open_Score

```
-- Valid_Open_Score : Il punteggio dato alla risposta aperta, deve essere compreso tra maxScore e minScore.
CREATE OR REPLACE FUNCTION VOS_function() RETURNS TRIGGER AS $Valid_Open_Score$
DECLARE
  min OPEN_QUIZ.MinScore%TYPE;
  max OPEN_QUIZ.MaxScore%TYPE;
BEGIN

  -- Prendo il minimo ed il massimo
  SELECT MinScore, MaxScore INTO min, max
  FROM OPEN_QUIZ
  WHERE CodOQ = NEW.CodOQ;

  IF NEW.Score NOT BETWEEN min AND max THEN
    RAISE EXCEPTION USING ERRCODE='SNIMM';
  END IF;

  RETURN NEW;

EXCEPTION
  WHEN SQLSTATE 'SNIMM' THEN
    UPDATE OPEN_ANSWER SET Score = 0 WHERE CodOA = NEW.CodOA;
    RAISE NOTICE 'Il punteggio deve essere compreso tra i valori fissati!';
    RETURN NULL;
END; $Valid_Open_Score$ LANGUAGE PLPGSQL;

CREATE OR REPLACE TRIGGER Valid_Open_Score
AFTER UPDATE ON OPEN_ANSWER
FOR EACH ROW
EXECUTE PROCEDURE VOS_function();
```

### 5.8.7 Unique_Username e Unique_Email

```
-- Unique_Username : Non devono esistere pi  utenti con lo stesso username
-- Unique_Email : Non devono esistere pi  utenti con la stessa email

CREATE OR REPLACE FUNCTION UU_function() RETURNS TRIGGER AS $$
DECLARE
  stmt VARCHAR(100);
  stmtE VARCHAR(100);
  stmtU VARCHAR(100);
  isEmail INT := 0;
  isUsername INT := 0;
  tab VARCHAR(20);
BEGIN
  -- A seconda della tabella in cui ho inserito, prendo la duale
  IF TG_TABLE_NAME = 'professor' THEN tab := 'STUDENT'; END IF;
  IF TG_TABLE_NAME = 'student' THEN tab := 'PROFESSOR'; END IF;

  -- Cerco se esiste una riga nella duale
  stmtE := concat('SELECT COUNT(*) FROM ',tab,' WHERE Email = $1;');
  stmtU := concat('SELECT COUNT(*) FROM ',tab,' WHERE Username = $1;');

  IF stmtE IS NOT NULL THEN EXECUTE stmtE INTO isEMail USING NEW.Email; END IF;
  IF stmtU IS NOT NULL THEN EXECUTE stmtU INTO isUsername USING New.Username; END IF;

  -- Elimino dalla tabella chiamante
  IF isEmail = 1 THEN
    stmt := concat('DELETE FROM ',TG_TABLE_NAME,' AS T WHERE T.Email = $1 ;');
    EXECUTE stmt USING NEW.Email;
    RAISE NOTICE 'Email gi  utilizzata da un altro utente!';

  ELSEIF isUsername = 1 THEN
    stmt := concat('DELETE FROM ',TG_TABLE_NAME,' AS T WHERE T.Username = $1 ;');
    EXECUTE stmt USING NEW.Username;
    RAISE NOTICE 'Username gi  esistente!';

  END IF;

  RETURN NULL;
END; $$ LANGUAGE PLPGSQL;
```

```
CREATE OR REPLACE TRIGGER unique_student_username AFTER INSERT ON
    STUDENT
FOR EACH ROW
EXECUTE PROCEDURE UU_function();

CREATE OR REPLACE TRIGGER unique_professor_username AFTER INSERT ON
     PROFESSOR
FOR EACH ROW
EXECUTE PROCEDURE UU_function();
```

### 5.8.8 Username_for_student

```
-- Procedura per il cambio di username di uno studente
CREATE OR REPLACE PROCEDURE username_for_student(s_user STUDENT.
    Username%TYPE, s_id STUDENT.StudentID%TYPE) AS $$
DECLARE
  info INT;
BEGIN
  --START TRANSACTION;
    RAISE NOTICE 'Transazione iniziata';
    SELECT COUNT(*) INTO info FROM PROFESSOR WHERE username =
    s_user;

    IF info = 1 THEN
      RAISE EXCEPTION USING ERRCODE='UNALP';  -- Username already
    exists in professor
    END IF;

    UPDATE STUDENT SET username = s_user WHERE StudentID = s_id;
  COMMIT;
  RAISE NOTICE 'Transazione finita';
EXCEPTION
  WHEN SQLSTATE 'UNALP' THEN
    RAISE NOTICE 'Esiste gi  un altro utente con quell username,
    eseguito un rollback';
    ROLLBACK;
  WHEN OTHERS THEN
    RAISE NOTICE 'Errore durante la transazione, eseguito un
    rollback';
    RAISE NOTICE 'SQLSTATE = %', SQLSTATE;
    ROLLBACK;
END; $$ LANGUAGE PLPGSQL;
```

### 5.8.9 Email_for_student

```
-- Procedura per il cambio di email di uno studente
CREATE OR REPLACE PROCEDURE email_for_student(s_email STUDENT.
    Username%TYPE, s_id STUDENT.StudentID%TYPE) AS $$
DECLARE
  info INT;
BEGIN
  --START TRANSACTION;
    RAISE NOTICE 'Transazione iniziata';
    SELECT COUNT(*) INTO info FROM PROFESSOR WHERE email = s_email;

```

```
    IF info = 1 THEN
      RAISE EXCEPTION USING ERRCODE='EMALP';  -- email already exists in professor
    END IF;

    UPDATE STUDENT SET email = s_email WHERE StudentID = s_id;
  COMMIT;
  RAISE NOTICE 'Transazione finita';
EXCEPTION
  WHEN SQLSTATE 'EMALP' THEN
    RAISE NOTICE 'Esiste gi  un altro utente con quell email';
    ROLLBACK;
  WHEN OTHERS THEN
    RAISE NOTICE 'Errore durante la transazione, eseguito un rollback';
    RAISE NOTICE 'SQLSTATE = %', SQLSTATE;
    ROLLBACK;
END; $$ LANGUAGE PLPGSQL;
```

### 5.8.10 Username_for_professor

```
-- Procedura per il cambio di username di un professore
CREATE OR REPLACE PROCEDURE username_for_professor(p_user PROFESSOR.Username%TYPE, p_id PROFESSOR.CodP%TYPE) AS $$
DECLARE
  info INT;
BEGIN
  --START TRANSACTION;
    RAISE NOTICE 'Transazione iniziata';
    SELECT COUNT(*) INTO info FROM STUDENT WHERE username = p_user;

    IF info = 1 THEN
      RAISE EXCEPTION USING ERRCODE='UNALS';  -- Username already exists in student
    END IF;

    UPDATE PROFESSOR SET username = p_user WHERE CodP = p_id;
  COMMIT;
  RAISE NOTICE 'Transazione finita';
EXCEPTION
  WHEN SQLSTATE 'UNALS' THEN
    RAISE NOTICE 'Esiste gi  un altro utente con quell username';
    ROLLBACK;
  WHEN OTHERS THEN
    RAISE NOTICE 'Errore durante la transazione, eseguito un rollback';
    RAISE NOTICE 'SQLSTATE = %', SQLSTATE;
    ROLLBACK;
END; $$ LANGUAGE PLPGSQL;
```

### 5.8.11 Email_for_professor

```
-- Procedura per il cambio di email di un professore
CREATE OR REPLACE PROCEDURE email_for_professor(p_email PROFESSOR.
    Username%TYPE, p_id PROFESSOR.CodP%TYPE) AS $$
DECLARE
  info INT;
BEGIN
  --START TRANSACTION;
    RAISE NOTICE 'Transazione iniziata';
    SELECT COUNT(*) INTO info FROM STUDENT WHERE email = p_email;

    IF info = 1 THEN
      RAISE EXCEPTION USING ERRCODE='EMALS';  -- email already
    exists in student
    END IF;

    UPDATE PROFESSOR SET email = s_email WHERE CodP = p_id;
  COMMIT;
  RAISE NOTICE 'Transazione finita';
EXCEPTION
  WHEN SQLSTATE 'EMALS' THEN
    RAISE NOTICE 'Esiste gi  un altro utente con quell email';
    ROLLBACK;
  WHEN OTHERS THEN
    RAISE NOTICE 'Errore durante la transazione, eseguito un
    rollback';
    RAISE NOTICE 'SQLSTATE = %', SQLSTATE;
    ROLLBACK;
END; $$ LANGUAGE PLPGSQL;
```

### 5.8.12 Revise_function

```
-- revise_function : procedura che inserito un test, ed un
    professore, "committa" tutti i test_taken, di un determinato
    test.
-- Verr  usata nell'applicativo

CREATE OR REPLACE PROCEDURE
revise_function(ctest TEST.CodTest%TYPE, cprof PROFESSOR.CodP%TYPE)
     AS $$
DECLARE rowtest TEST%ROWTYPE;
BEGIN
  SELECT * INTO rowtest FROM TEST WHERE CodTest = ctest;
  IF rowtest.CodP = cprof THEN
    UPDATE TEST_TAKEN SET Passed = true WHERE CodTest = ctest;
  ELSE
    RAISE NOTICE 'Hai inserito un test non tuo';
  END IF;
END; $$ LANGUAGE PLPGSQL;
```

### 5.8.13 getStudent

```
-- getStudent: dato in input lo Username di uno studente,
    restituisce lo StudentID di quello studente

CREATE OR REPLACE FUNCTION
getStudent(studUsername STUDENT.Username%TYPE)
RETURNS STUDENT.StudentID%TYPE
AS $$
BEGIN
  RETURN (SELECT StudentID
      FROM STUDENT
      WHERE studUsername = STUDENT.Username);
END; $$ LANGUAGE PLPGSQL;
```

### 5.8.14 getProfessor

```
-- getProfessor: dato in input lo Username di un professore,
-- restituisce il CodP di quel professore

CREATE OR REPLACE FUNCTION
getProfessor(profUsername PROFESSOR.Username%TYPE)
RETURNS PROFESSOR.CodP%TYPE
AS $$
BEGIN
  RETURN (SELECT CodP
      FROM PROFESSOR
      WHERE profUsername = PROFESSOR.Username);
END; $$ LANGUAGE PLPGSQL;
```

### 5.8.15 getTest

```
-- getTest: dato in input il nome di un test, restituisce il Codice
    di quel Test
CREATE OR REPLACE FUNCTION
getTest(testName TEST.Name%TYPE)
RETURNS TEST.CodTest%TYPE
AS $$
BEGIN
  RETURN (SELECT CodTest
      FROM TEST
      WHERE testName = TEST.Name);
END; $$ LANGUAGE PLPGSQL;
```

### 5.8.16 getClass

```
-- getClass: dato in input il nome di un corso, restituisce il CodC
    di quel corso
CREATE OR REPLACE FUNCTION
getClass(className CLASS_T.Name%TYPE)
RETURNS CLASS_T.CodC%TYPE
AS $$
BEGIN
  RETURN (SELECT CodC
      FROM CLASS_T
      WHERE className = CLASS_T.Name);
END; $$ LANGUAGE PLPGSQL;
```

### 5.8.17 getLecture

```
-- getTest: dato in input il titolo di una lezione, restituisce il
    CodL di quella lezione
CREATE OR REPLACE FUNCTION
getLecture(lecTitle LECTURE.Title%TYPE)
RETURNS LECTURE.CodL%TYPE
AS $$
BEGIN
  RETURN (SELECT CodL
      FROM LECTURE
      WHERE lecTitle = LECTURE.Title);
END; $$ LANGUAGE PLPGSQL;
```

## 5.9 Popolazione

Per un esempio di popolazione seguire questo link.