



**MANABI**  
**Applicazione E-learning tramite quiz**  
**Traccia 2**

Giorgio Longobardo N86003571  
Domitilla Giulia Simeoli N86003780  
Anno Accademico 2021/2022

Docente:  
Silvio Barra

# Indice

0.1	Descrizione del progetto . . . . .	4
0.2	Requisiti identificati . . . . .	4
<b>1</b>	<b>Progettazione concettuale</b>	<b>5</b>
1.1	Class Diagram . . . . .	6
1.2	Analisi della ristrutturazione del Class Diagram . . . . .	7
1.2.1	Analisi delle ridondanze . . . . .	7
1.2.2	Analisi degli identificativi . . . . .	7
1.2.3	Rimozione degli attributi multipli . . . . .	7
1.2.4	Rimozione degli attributi composti . . . . .	7
1.2.5	Partizione/Accorpamento delle associazioni . . . . .	7
1.2.6	Rimozione delle gerarchie . . . . .	7
1.3	Class Diagram ristrutturato . . . . .	8
1.4	Dizionario delle classi . . . . .	9
1.5	Dizionario delle associazioni . . . . .	10
1.6	Dizionario dei vincoli . . . . .	11
<b>2</b>	<b>Progettazione logica</b>	<b>12</b>
2.1	Schema logico . . . . .	12
<b>3</b>	<b>Progettazione fisica e definizioni SQL</b>	<b>13</b>
3.1	Introduzione . . . . .	13
3.2	Procedure individuate . . . . .	13
3.3	Automazioni . . . . .	13
3.4	Valori Default . . . . .	13
3.5	Viste implementate . . . . .	14
3.6	Definizioni tabelle . . . . .	15
3.6.1	Insegnante . . . . .	15
3.6.2	Studiante . . . . .	15
3.6.3	Test . . . . .	15
3.6.4	Correzione . . . . .	15
3.6.5	Quesito Aperto . . . . .	16
3.6.6	Risposta Aperta . . . . .	16
3.6.7	Quesito Multiplo . . . . .	16
3.6.8	Risposte Multiple . . . . .	16
3.7	Implementazione vincoli sugli attributi di un'ennupla . . . . .	17
3.7.1	Controllo Punteggio Aperto . . . . .	17
3.7.2	Controllo Punteggio Multiplo . . . . .	17
3.7.3	Controllo Tempo . . . . .	17
3.7.4	Controllo credenziali utente . . . . .	17
3.7.5	Controllo password utente . . . . .	18
3.8	Implementazione vincoli Interrelazionali . . . . .	20
3.8.1	Controllo punteggio Assegnato . . . . .	20
3.8.2	Controllo Valore booleano . . . . .	20
3.8.3	Controllo Dominio risposta Multipla . . . . .	21
3.8.4	Controllo username . . . . .	22
3.8.5	Controllo creatore del test . . . . .	23
3.9	Implementazione Automazioni . . . . .	23
3.9.1	Aggiornamento voto . . . . .	23

3.9.2	Aggiornamento punteggio totale . . . . .	25
3.10	Implementazioni viste . . . . .	25
3.10.1	Visualizza dati studente . . . . .	25
3.10.2	Risposte esatte per studente . . . . .	25
3.10.3	Risposte errate per studente . . . . .	26
3.10.4	Visualizza risultati test . . . . .	26
3.10.5	Visualizza media per materia . . . . .	26

## 0.1 Descrizione del progetto

*Manabi* è un applicativo e-learning il cui intento è quello di gestire test. Questi possono essere inseriti da un insegnante, dopo una sua registrazione inserendo credenziali univoche, e ogni test è costituito da un insieme di quesiti, distinti in due categorie: a risposta 'multipla', le quali prevedono più possibili risposte, di cui una sola è da ritenersi corretta, e a risposta 'aperta', alle quali si può rispondere con un testo apposito. Ad entrambi è previsto un punteggio da assegnare in caso di risposta giusta o sbagliata, con l'eccezione che le risposte 'aperte' sono destinate alla valutazione dell'insegnante che provvederà ad assegnare il relativo punteggio in base alla correttezza della risposta. I test sono destinati agli studenti che si registrano all'applicativo, dotati di credenziali uniche, e, al termine della correzione, sono in grado di visualizzare il punteggio ottenuto dei vari test compilati.

## 0.2 Requisiti identificati

I requisiti da soddisfare sono stati essenzialmente tre: l'interazione fra insegnante e test, l'interazione fra studente e test e infine l'interazione fra test e risposte. Innanzitutto, abbiamo distinto l'utente in due categorie: un utente insegnante ed un utente studente. Entrambi hanno come chiave primaria una stringa che individua il loro username e svolgono due ruoli diversi nella Base di Dati: l'insegnante provvede a creare, modificare e correggere i test, mentre uno studente è tenuto solo a svolgerli e a visualizzare i risultati ottenuti. Si è esclusa l'idea di rendere l'utente come un'unica entità per due motivi principali:

- Alleggerimento della Base di Dati: una eventuale tabella unica denominata *Utente* avrebbe dovuto condividere attributi in comune che non sarebbero stati utilizzati nel caso la tabella sarebbe stata letta come "insegnante" o "studente": ci sarebbero stati molti valori null (come, ad esempio, nel caso di eventuale attributo *punteggio* se *Utente* sarebbe da considerare come insegnante).
- Maggiore chiarezza: un eventuale tabella unica avrebbe reso le idee confuse durante l'interpretazione. In che modo e quando *Utente* si sarebbe dovuto leggere come insegnante o come studente? Inserire un nuovo attributo (magari enumerativo o booleano) avrebbe sì chiarito l'interpretazione, ma sarebbe stato difficoltoso distinguere ogni volta quel valore durante le query.
- Ridotto costo computazionale: se avessimo inserito un nuovo attributo all'eventuale tabella *Utente* che identificasse con che tipo di utente si sta interagendo, si sarebbero dovuti svolgere ulteriori controlli sull'attributo per ogni comando, andando così a incrementare il costo computazionale ad ogni query.

I quiz sono rappresentati nella Base di Dati con due tabelle differenti, chiamate *Quiz Aperto* e *Quiz Multiplo*. Le due tabelle hanno una chiave primaria che identifica univocamente la domanda e una chiave esterna che fa riferimento al test di appartenenza. Inoltre il quiz multiplo ha un attributo multiplo *risposte* che contiene le eventuali risposte proposte allo studente da scegliere. Abbiamo anche qui escluso l'idea di creare un'unica entità *Quiz* che potesse contenere entrambi i tipi di quiz per i seguenti motivi:

- Rimozione di attributi inutili: una sola eventuale entità *Quiz* avrebbe dovuto contenere anche le risposte proposte a quella domanda. Se il quiz indicato fosse stato a domanda aperta, allora tutte le risposte proposte dovrebbero essere poste a null, andando a riempire la base di dati con elementi fittizi.
- Costo computazionale: anche qui si sarebbe dovuto verificare ad ogni query il tipo di quiz, incrementando dunque il costo per ogni operazione effettuata sul database.

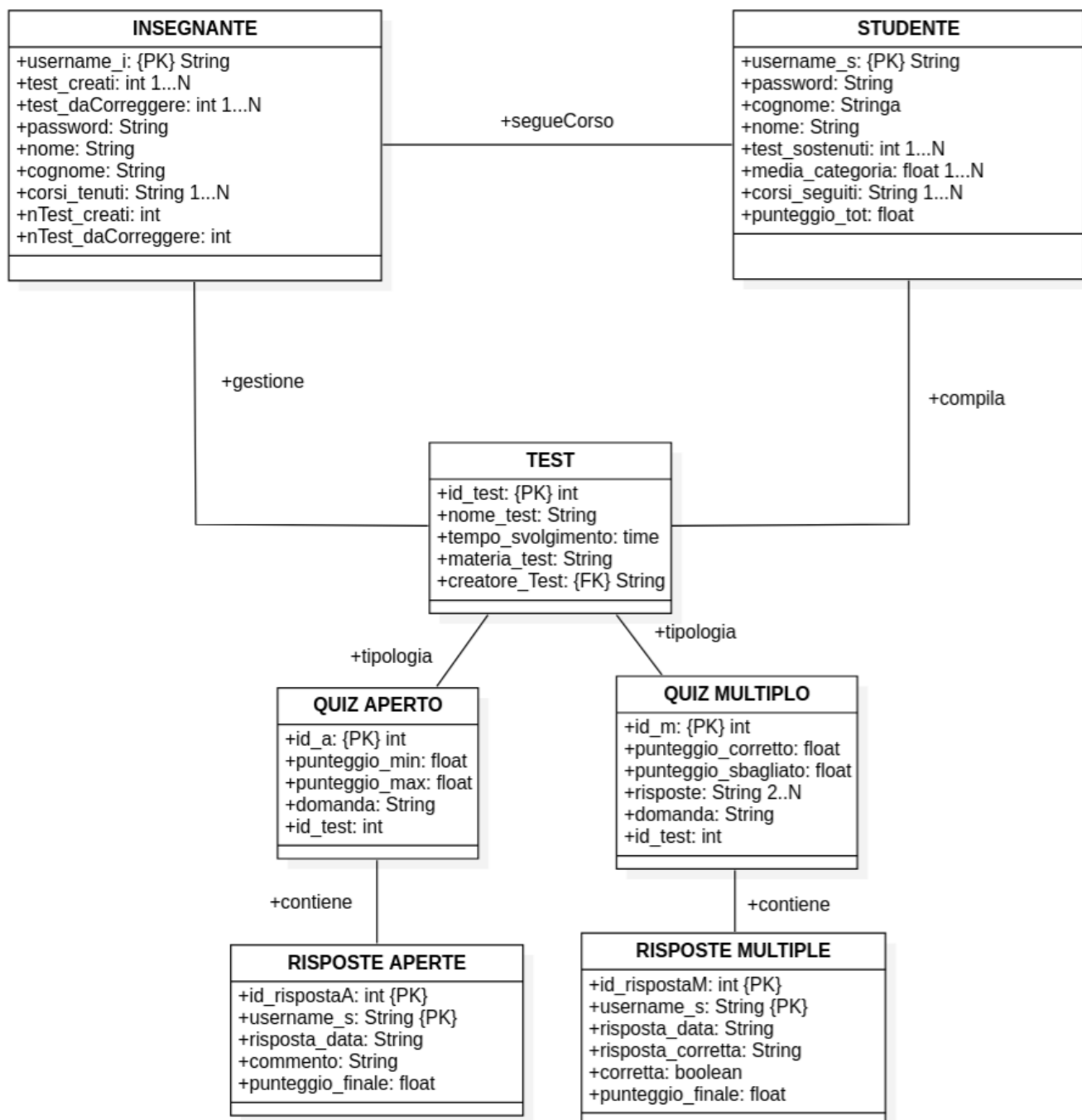
Infine, per memorizzare le risposte si è deciso di creare due tabelle di supporto, *Risposte Aperte* e *Risposte Multiple*: la prima contiene due chiavi esterne che si riferiscono all'username dello studente e alla chiave primaria del quiz e insieme formano la chiave primaria nella tabella, la seconda invece contiene, oltre alle due chiavi primarie sopracitate, anche una chiave esterna che si riferisce ad una risposta contenuta nell'attributo multiplo *Risposte* di *Quiz Multiplo* in base alla risposta data dallo studente.



# Capitolo 1

## Progettazione concettuale

### 1.1 Class Diagram



## 1.2 Analisi della ristrutturazione del Class Diagram

Si prosegue ora all'analisi della ristrutturazione del Class Diagram. I parametri da controllare sono le ridondanze, identificativi, attributi multipli, attributi composti, partizioni o accorpamenti delle associazioni e infine rimozioni delle eventuali gerarchie.

### 1.2.1 Analisi delle ridondanze

Il primo attributo ridondante che abbiamo identificato è *Corsi Tenuti* di Insegnante: poiché ogni Test possiede una rispettiva materia, si possono dedurre i corsi impartiti dal professore mediante i test da lui creati. Altri attributi ridondanti della tabella insegnante sono *nTest creati* e *nTest da Correggere*. Essi sono usati per identificare la quantità di test creati da un singolo professore, ma tale informazione è deducibile sommando i test creati e verificare in seguito quale di questi sono da correggere. Abbiamo deciso di creare una nuova tabella chiamata *Correzione*, la quale possiede le informazioni utili riguardante la relazione Insegnante-Test-Studente. Vedasi la sottosezione 'Analisi degli attributi multipli' per maggiori dettagli. Passando ora alla tabella studente, l'attributo ridondante identificato è *Media categoria*. Dato che è ricavabile facendo la media attraverso la tabella Test, abbiamo deciso di usare una vista per tenere traccia della media degli studenti. Non sono state rivelate ulteriori ridondanze.

### 1.2.2 Analisi degli identificativi

Nelle tabelle abbiamo tentato di usare identificativi autoesplicativi. Nel caso in cui sono presenti chiavi esterne nelle tabelle, esse sono state denominate come l'attributo a cui fanno riferimento. Inoltre, per maggior chiarezza, le tabelle *Quiz Multiplo* e *Quiz Aperto* sono state ridenominate in *Quesito Multiplo* e *Quesito Aperto* rispettivamente. Infine, abbiamo deciso di rimuovere il nome delle associazioni fra le tabelle per non confondere le idee al lettore.

### 1.2.3 Rimozione degli attributi multipli

Gli attributi multipli presenti nel class diagram sono molteplici. Nella tabella *Insegnante*, *Test Creati* tiene traccia di tutti i test creati da un professore, riferendosi alle chiavi primarie di un test. Dato che non è possibile implementare un attributo multiplo all'interno di un database, la scelta attuata è stata quella di creare una nuova tabella denominata *Correzione* ed essa possiede i seguenti attributi: *Username insegnante* che contiene l'username dell'insegnante, creatore del test, *id Test* che contiene l'identificativo del test creato dal professore, *Username studente*, che tiene traccia dello studente che ha svolto quel determinato test, *Corretto* che segnala la correzione di un test svolto dallo studente da parte del professore e infine *Voto Test* che salva il punteggio ottenuto di uno studente. Usando questa tabella, si possono rimuovere anche gli attributi *Test Sostenuti* e *Corsi Seguiti* dalla tabella *Studente*, dacché ricavabili dalla nuova tabella creata. Un ulteriore attributo multiplo da rimuovere è *risposte* nella tabella *Quesito Multiplo*, la quale contiene tutte le possibili risposte ad un determinato quiz. Giacché un quiz non può contenere infinite possibili risposte e bisogna identificare univocamente una risposta vera, abbiamo partizionato questo attributo in cinque attributi: uno che contiene la risposta esatta al quiz che lo contiene e altri quattro contenenti risposte false, tutte in alternativa alla prima domanda. Ci teniamo a precisare che la prima e una delle quattro non possono essere null, mentre le tre rimanenti possono esserlo, così da poter creare quesiti multipli aventi al massimo 5 possibili risposte.

### 1.2.4 Rimozione degli attributi composti

Non sono presenti attributi composti.

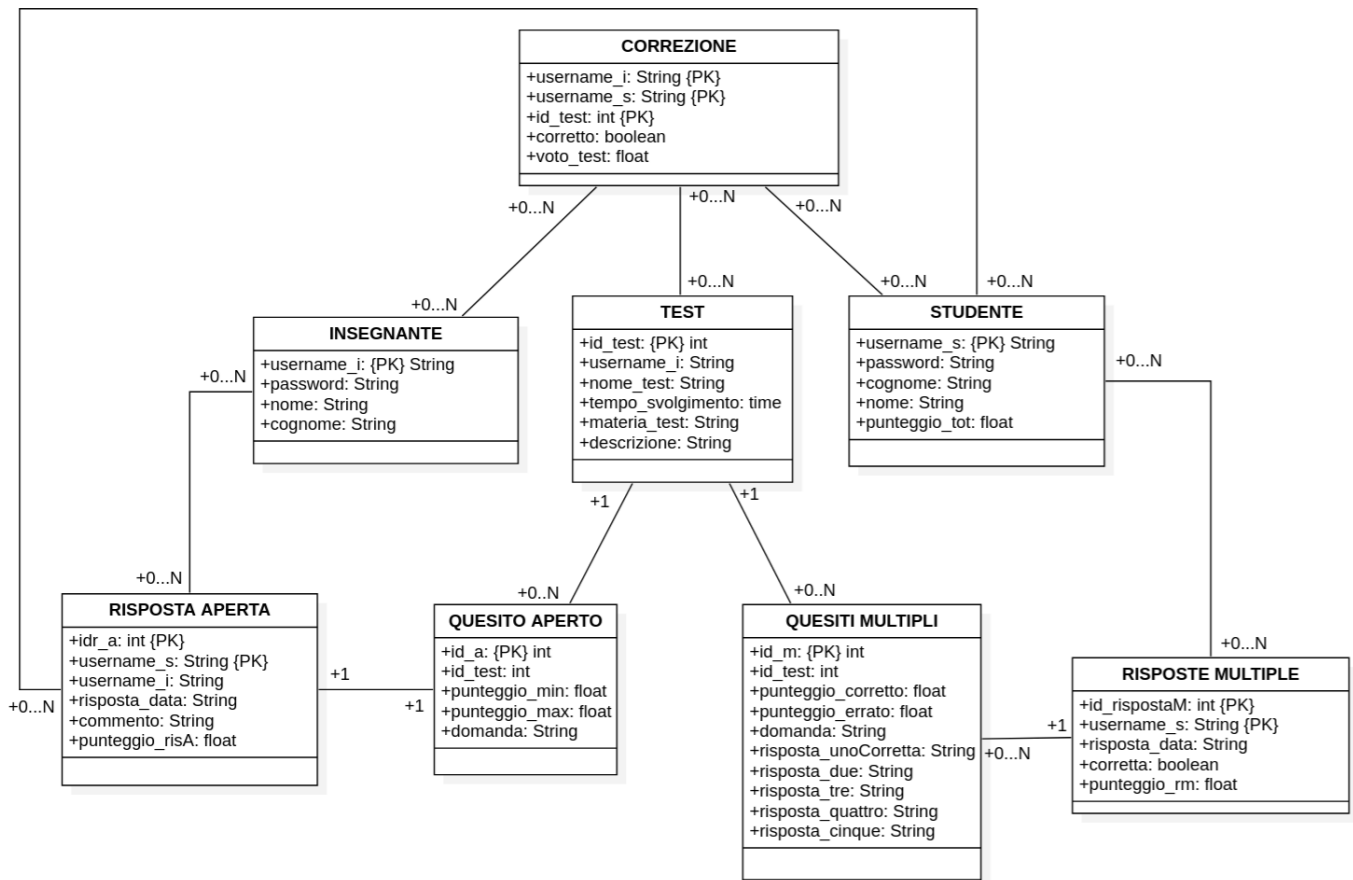
### 1.2.5 Partizione/Accorpamento delle associazioni

Come già spiegato nella sottosezione 'Analisi degli attributi multipli', le associazioni fra *Insegnante*, *Studente* e *Test* sono state racchiuse tutte nella singola tabella *Correzione*, che sarà poi collegata a *Test*. E' stata poi aggiunta una nuova associazione che collega l'username dell'insegnante ad una risposta aperta data da uno studente.

### 1.2.6 Rimozione delle gerarchie

Non sono presenti gerarchie.

## 1.3 Class Diagram ristrutturato





## 1.4 Dizionario delle classi

<i>Classe</i>	<i>Descrizione</i>	<i>Attributi</i>
<b>Insegnante</b>	Descrittore dell'utente insegnante	<b>Username_i (String)</b> : Chiave primaria. Identifica univocamente un utente insegnante. <b>Password_i (String)</b> : Password associata all'username. <b>Nome (String)</b> : Nome associato all'insegnante. <b>Cognome (String)</b> : Cognome associato all'insegnante.
<b>Studente</b>	Descrittore dell'utente studente	<b>Username_s (String)</b> : Chiave primaria. Identifica univocamente un utente studente. <b>Password_s (String)</b> : Password associata all'username. <b>Nome (String)</b> : Nome associato allo studente. <b>Cognome (String)</b> : Cognome associato allo studente. <b>Punteggio_tot (String)</b> : Punteggio totale assegnato ad uno studente.
<b>Test</b>	Descrittore di un test	<b>id_Test (int)</b> : Chiave primaria. Identifica univocamente un singolo test. <b>Username_i (String)</b> : Chiave esterna. Identifica il creatore del test. <b>Nome_Test (String)</b> : Nome o titolo assegnato al test. <b>Tempo_Svolgimento (Time)</b> : Tempo necessario per lo svolgimento. <b>Materia_Test (String)</b> : Materia associata al test. <b>Descrizione (String)</b> : Eventuale descrizione del test.
<b>Correzione</b>	Descrittore di una classe deputata al mantenimento delle relazioni fra insegnante, studente e test	<b>Username_i (String)</b> : Username dell'insegnante. Deve essere obbligatoriamente il creatore del test. <b>Username_s (String)</b> : Username dello studente. Deve essere inserito dopo aver partecipato ad un test. <b>id_Test (int)</b> : identificativo del test a cui uno studente partecipa. Insieme a username_s e username_i formano la chiave primaria. <b>corretto (boolean)</b> : Segnala se il determinato test effettuato da uno specifico studente sia stato corretto. <b>Voto_Test (float)</b> : punteggio del test effettuato da uno studente.
<b>Quesito Aperto</b>	Descrittore della classe contenente il tipo di quiz a risposta aperta	<b>id_a (int)</b> : Chiave primaria. Identifica il quiz univocamente. <b>id_Test (int)</b> : Chiave esterna. Fa riferimento al test che lo contiene. <b>punteggio_min (float)</b> : Punteggio minimo possibile alla domanda. <b>punteggio_max (float)</b> : Punteggio massimo possibile alla domanda. <b>domanda (String)</b> : La domanda da porre allo studente.
<b>Quesito Multiplo</b>	Descrittore della classe contenente il tipo di quiz a risposta multipla.	<b>id_m (int)</b> : Chiave primaria. Identifica il quiz univocamente. <b>id_Test (int)</b> : Chiave esterna. Fa riferimento al test che lo contiene. <b>punteggio_corretto (float)</b> : Punteggio da assegnare in caso di risposta esatta. <b>punteggio_errato (float)</b> : Punteggio da assegnare in caso di risposta errata. <b>domanda (String)</b> : Domanda da porre allo studente. <b>risposta_unoCorretta (String)</b> : Prima risposta. E' quella corretta. <b>risposta_due (String)</b> : Seconda risposta. <b>risposta_tre (String)</b> : Terza eventuale risposta. <b>risposta_quattro (String)</b> : Quarta eventuale risposta. <b>risposta_cinque (String)</b> : Quinta eventuale risposta.
<b>Risposta Aperta</b>	Descrittore della classe contenente le risposte aperte degli studenti.	<b>idr_a (int)</b> : Chiave esterna. Fa riferimento all'id del quiz. <b>Username_s (String)</b> : Chiave esterna. Fa riferimento allo studente che ha risposto. Insieme a idr_a forma la chiave primaria. <b>Username_i (String)</b> : Chiave esterna. Fa riferimento all'insegnante che deve correggere la risposta. <b>risposta_data (String)</b> : Testo contenente la risposta dello studente. <b>commento (String)</b> : Eventuale commento dell'insegnante. <b>punteggio_risposta (float)</b> : Punteggio assegnato dall'insegnante.
<b>Risposta Multipla</b>	Descrittore della classe contenente le risposte multiple degli studenti.	<b>id_risposta (int)</b> : Chiave esterna. Fa riferimento all'id del quiz. <b>Username_s (String)</b> : Chiave esterna. Fa riferimento allo studente che ha risposto. Insieme a id_risposta forma la chiave primaria. <b>risposta_data (String)</b> : Testo contenente la risposta dell'utente. Deve essere una delle possibili risposte proposte. <b>corretta (boolean)</b> : Valore booleano che indica o meno la correttezza della risposta. <b>punteggio_risposta(float)</b> : Punteggio relativo alla domanda. Deve essere un punteggio valido.

## 1.5 Dizionario delle associazioni

Viene di seguito riportato il dizionario delle associazioni: i nomi non sono presenti nel Class Diagram per maggior leggibilità, mentre qui sono riportate tutte le informazioni utili all'interpretazione.

<i>Associazione</i>	<i>Tabelle coinvolte</i>
<b>Correzione Test</b>	<b>Insegnante</b> [*] ruolo (corregge): indica quale test e quali studenti deve correggere <b>Correzione</b> [*] ruolo (coinvolge): indica tutti i professori che devono effettuare correzioni.
<b>Creazione Test</b>	<b>Insegnante</b> [*] ruolo (creatore): indica quale test crea un professore <b>Test</b> [*] ruolo (creato): indica il proprietario del test
<b>Correzione Risposte</b>	<b>Insegnante</b> [*] ruolo (corregge): indica quale risposta deve correggere un professore. <b>Risposta Aperta</b> [*] ruolo (corretta): indica la risposta corretta dal professore
<b>Correzione Test studente</b>	<b>Studente</b> [*] ruolo (corretto): indica quali test ha svolto uno studente. <b>Correzione</b> [*] ruolo (coinvolge): indica gli studenti coinvoltinella correzione.
<b>Assegna Risposta aperta</b>	<b>Studente</b> [*] ruolo (assegna): indica quale studente assegna la risposta <b>Risposta Aperta</b> [*] ruolo (assegnata): indica la risposta assegnata da uno studente.
<b>Assegna Risposta multipla</b>	<b>Studente</b> [*] ruolo (assegna): indica quale studente assegna la risposta. <b>Risposte Multiple</b> [*] ruolo (assegnata): indica la risposta assegnata da uno studente.
<b>Correlazione Test</b>	<b>Correzione</b> [*] ruolo (contiene): indica quale insegnante deve correggere il test e quale studente ha svolto il test. <b>Test</b> [*] ruolo (identifica): indica quale test deve essere corretto dal professore.
<b>Contenenza Quesiti aperti</b>	<b>Test</b> [1] ruolo (contiene): indica quale test contiene i quiz. aperti. <b>Quesiti Aperti</b> [*] ruolo (contenuto): indica quale quiz è contenuto nel test.
<b>Contenenza Quesiti multipli</b>	<b>Test</b> [1] ruolo (contiene): indica quale test contiene i quiz multipli. <b>Quesiti Mulitpli</b> [*] ruolo (contenuto): indica quale quiz è contenuto nel test.
<b>Contenenza Risposte aperte</b>	<b>Quesiti Aperti</b> [*] ruolo (contiene): indica quale quiz contiene le risposte. <b>Risposta Aperta</b> [1] ruolo (contenuta): indica quale risposta aperta è contenuta in un quiz.
<b>Contenenza Risposte multiple</b>	<b>Quesiti Multipli</b> [*] ruolo (contiene): indica quale quiz contiene le risposte. <b>Risposte Multiple</b> [1] ruolo (contenute): indica quale risposta multipla è contenuta in un quiz.

## 1.6 Dizionario dei vincoli

Viene di seguito riportato il dizionario dei vincoli. Viene specificato il nome, il tipo e una descrizione.

<i>Vincoli</i>	<i>Tipo</i>	<i>Descrizione</i>
<b>Controllo Punteggio Aperto</b>	<b>N-upla</b>	Vieta la possibilità in 'Quesito Aperto' di un punteggio minimo maggiore del punteggio massimo.
<b>Controllo Punteggio Multiplo</b>	<b>N-upla</b>	Vieta la possibilità in 'Quesito Multiplo' di un punteggio esatto minore al punteggio errato e inoltre stabilisce che il primo deve essere maggiore stretto di 0 mentre il secondo minore o uguale a 0.
<b>Controllo Tempo</b>	<b>N-upla</b>	Controlla che il tempo disponibile per un test sia o null (tempo illimitato) o almeno di 15 minuti.
<b>Controllo caratteri minimi username</b>	<b>N-upla</b>	Stabilisce che l'username di un insegnante o di uno studente deve essere composto da almeno 8 caratteri.
<b>Controllo nome e cognome</b>	<b>N-upla</b>	Stabilisce che il nome e il cognome di un insegnante e di uno studente deve essere lungo almeno 2 caratteri. Inoltre, non possono esserci utenti con nome e cognome uguali.
<b>Controllo password</b>	<b>N-upla</b>	Stabilisce che la password di un utente debba avere lunghezza minima di 8 caratteri.
<b>Validità password</b>	<b>N-upla</b>	Stabilisce che una password non debba essere null.
<b>Validità nome test</b>	<b>N-upla</b>	Stabilisce che un test non debba essere null.
<b>Validità creatore test</b>	<b>N-upla</b>	Stabilisce che un test debba avere un creatore.
<b>Validità quiz aperto</b>	<b>N-upla</b>	Stabilisce che un quesito aperto abbia una domanda, un punteggio minimo e un punteggio massimo non null.
<b>Validità quiz multiplo</b>	<b>N-upla</b>	Stabilisce che un quesito multiplo abbia almeno due domande, inoltre che abbia un punteggio corretto ed errato non nulli.
<b>Unicità delle chiavi primarie</b>	<b>Intrarelazionale</b>	Stabilisce l'unicità delle chiavi primarie definite nelle tabelle.
<b>Controllo punteggio Assegnato</b>	<b>Interrelazionale</b>	Stabilisce che il punteggio assegnato dal professore ad una risposta aperta sia un valore compreso fra il minimo e il massimo possibile
<b>Controllo Voto</b>	<b>Interrelazionale</b>	Stabilisce che il voto in 'Correzione' sia la somma dei punteggi ottenuti dalle risposte aperte e multiple di uno studente.
<b>Controllo valore booleano</b>	<b>Interrelazionale</b>	Controlla che il valore booleano in 'Risposte Multiple' sia corretto, in base alla veridicità di una risposta assegnata da uno studente.
<b>Controllo dominio risposta mutlipa</b>	<b>Interrelazionale</b>	Controlla che la risposta assegnata da uno studente in 'Risposte Multiple' sia una risposta possibile proposta dal sistema.
<b>Controllo username</b>	<b>Interrelazionale</b>	Stabilisce che insegnanti e studente non debbano avere username identici.
<b>Controllo creatore del test</b>	<b>Interrelazionale</b>	Stabilisce che un professore può correggere solo test da lui creati.

## Capitolo 2

# Progettazione logica

Si prosegue ora con lo schema logico del class diagram. In grassetto sono evidenziate le chiavi primarie, mentre saranno sottolineate le chiavi esterne.

### 2.1 Schema logico

Insegnante	( <u>Username_i</u> , Password_i, Nome, Cognome)
Studente	( <u>Username_s</u> , Password_s, Nome, Cognome, Punteggio_Tot)
Test	( <u>Id_Test</u> , <u>Username_i</u> , Nome_Test, Tempo_Svolgimento, Materia_Test, Descrizione) Username_i $\rightarrow$ <i>Insegnante.Username_i</i>
Correzione	( <u>Username_i</u> , <u>Username_s</u> , <u>Id_Test</u> , Corretto, Voto_Test) Username_i $\rightarrow$ <i>Insegnante.Username_i</i> Username_s $\rightarrow$ <i>Studente.Username_s</i> Id_Test $\rightarrow$ <i>Test.Id_Test</i>
Quesito Aperto	( <u>Id_A</u> , <u>Id_Test</u> , Punteggio_Min, Punteggio_Max, Domanda) Id_Test $\rightarrow$ <i>Test.Id_Test</i>
Risposta Aperta	( <u>Idr_a</u> , <u>Username_s</u> , <u>Username_i</u> , Risposta_Data, Commento, Punteggio_Risa) Idr_a $\rightarrow$ <i>Quesito_Aperto.Id_A</i> Username_s $\rightarrow$ <i>Studente.Username_s</i> Username_i $\rightarrow$ <i>Insegnante.Username_i</i>
Quesito Multiplo	( <u>Id_M</u> , <u>Id_Test</u> , Punteggio_Corretto, Punteggio_Errato, Domanda, Risposta_UnoCorretta, Risposta_Due, Risposta_Tre, Risposta_Quattro, Risposta_Cinque) Id_Test $\rightarrow$ <i>Test.Id_Test</i>
Risposte Multiple	( <u>Idr_m</u> , <u>Username_s</u> , Risposta_Data, Corretta, Punteggio_Rism) Idr_m $\rightarrow$ <i>Quesito_Multiplo.Id_M</i> Username_s $\rightarrow$ <i>Studente.Username_s</i>

## Capitolo 3

# Progettazione fisica e definizioni SQL

### 3.1 Introduzione

Si passa ora alle definizioni delle tabelle, alle implementazioni dei vincoli e dei trigger. Il DBMS usato in esempio è *PostgreSQL 10*. Si rende noto che tutte le tabelle e tutti gli attributi che contenevano degli spazi nei nomi sono stati ridenominati inserendo un underscore al posto degli spazi. Quindi, ad esempio, la tabella *Quesito Aperto* nel Database sarà d'ora in poi chiamato come *Quesito\_Aperto*. Inoltre, nella tabella *Quesito Multiplo*, gli attributi che specificano il numero della risposta sono stati ridotti in questo modo: *Risposta\_UnoCorretta*  $\rightarrow$  *R\_UnoC*, *Risposta\_Due*  $\rightarrow$  *R\_Due*, ... e così via.

### 3.2 Procedure individuate

La maggior parte delle procedure individuate sono state perlopiù utilizzate per la corretta implementazione dei trigger a causa delle restrizioni di PostgreSQL. Sono state individuate le seguenti funzioni:

**Controllo\_Risa:** Controlla che il punteggio assegnato alla risposta aperta sia compreso fra il minimo e massimo possibile.

**Aggiorna\_voto\_a:** Somma a *Voto\_Test* il punteggio delle risposte aperte dato uno studente.

**Somma\_mul:** Somma a *Voto\_Test* il punteggio delle risposte multiple dato uno studente.

**Aggiorna\_PunteggiTot:** somma a *PunteggioTotdistudentetuttiivotidaluiconseguiti*.

**Controllo\_Tempo:** Controlla che il tempo di un test non sia minore di 15 minuti.

**Assegna\_bool:** Assegna il valore booleano a *Corretta* data la risposta dell'utente.

**Controllo\_Risposta:** Controlla che la risposta data dall'utente sia una delle risposte possibili.

**Controllo\_Username:** Controlla che gli username di studente e di insegnante non siano identici.

**Controllo\_lunghezza:** Controlla che le credenziali inserite dall'utente siano corrette.

**Verifica\_InsegnanteCorrezione:** verifica che l'insegnante corregga i test da lui creati.

Consultasi dalla sezione 4.6 per i dettagli.

### 3.3 Automazioni

Il DBMS fornisce le seguenti automazioni:

**Aggiorna Voto:** Somma a *Correzione.Voto\_Test* i punteggi relativi ad uno studente dopo aver inserito un punteggio.

**Aggiorna Punteggio Totale:** Somma a *Studente.Punteggio\_Tot* tutti i voti conseguiti di tutti i test.

**Assegna valore Booleano:** Stabilisce un valore booleano a *corretta* a *Risposte\_Multiple.Corretta* in base alla verifica della risposta data dello studente se coincide con la risposta corretta.

### 3.4 Valori Default

Inoltre, i seguenti attributi avranno di default i seguenti valori:

**Studente.Punteggio\_Tot** ha di default 0.

**Test.Descrizione** ha di default la stringa *Nessuna descrizione*.

**Correzione.Corretta** e **Correzione.Voto\_Test** avranno di default rispettivamente FALSE e 0.

**Risposta\_Aperta.Punteggio\_Risa** e **Risposta\_Aperta.Commento** avranno di default rispettivamente 0 e la stringa *Nessun commento*.

**Risposte\_Multiple.Punteggio\_Rism** avrà di default 0.

## 3.5 Viste implementate

Sono state implementate le seguenti liste:

**Visualizza\_Dati\_Studente:** essa ha come attributi *Nome\_Test* e *N\_Studenti*. Serve all'insegnante per vedere la partecipazione totale degli studenti.

**Risposte\_Esatte:** essa ha come attributi *Studente*, *Esatte*, *Test*. Serve per vedere la totalità di risposte esatte dati gli studenti ad un determinato test. Per le risposte aperte si controlla che il punteggio non sia il minimo.

**Risposte\_Errate:** essa ha come attributi *Studente*, *Errate*, *Test*. Serve per vedere la totalità di risposte sbagliate dati gli studenti ad un determinato test. Per le risposte aperte si controlla che il punteggio sia il minimo.

**Visualizza\_Risultati:** essa ha come attributi *Nome\_Stu*, *Nome\_Test*, *Nome\_Ins*, *Ris\_Esatte*, *Ris\_Errate*, *Voto*. Serve allo studente per vedere i risultati di un test. Fa uso delle due viste precedenti.

**Media\_Categoria:** essa ha come attributi *Nome\_Test*, *Materia*, *Studente*, *Media*. Serve ad uno studente per visualizzare la media data una categoria.

Consultasi la sezione 4.9 per i dettagli.

.

## 3.6 Definizioni tabelle

### 3.6.1 Insegnante

```
CREATE TABLE INSEGNANTE(  
    USERNAME_I VARCHAR(25) NOT NULL,  
    PASSWORD_I VARCHAR(25) NOT NULL,  
    NOME VARCHAR(25),  
    COGNOME VARCHAR(25),  
    PRIMARY KEY (USERNAME_I)  
);
```

### 3.6.2 Studente

```
CREATE TABLE STUDENTE(  
    USERNAME_S VARCHAR(25) NOT NULL,  
    PASSWORD_S VARCHAR(25) NOT NULL,  
    NOME VARCHAR(25),  
    COGNOME VARCHAR(25),  
    PUNTEGGIO_TOT FLOAT DEFAULT 0,  
    PRIMARY KEY (USERNAME_S)  
);
```

### 3.6.3 Test

```
CREATE TABLE TEST(  
    ID_TEST INT NOT NULL,  
    USERNAME_I VARCHAR(25) NOT NULL,  
    NOME_TEST VARCHAR(25) NOT NULL,  
    TEMPO_SVOLGIMENTO TIME,  
    MATERIA_TEST VARCHAR(25),  
    DESCRIZIONE VARCHAR(100) DEFAULT 'Nessuna descrizione.',  
    PRIMARY KEY (ID_TEST),  
    FOREIGN KEY (USERNAME_I) REFERENCES INSEGNANTE (USERNAME_I)  
);
```

### 3.6.4 Correzione

```
CREATE TABLE CORREZIONE(  
    USERNAME_I VARCHAR(25) NOT NULL,  
    USERNAME_S VARCHAR(25) NOT NULL,  
    ID_TEST INT NOT NULL,  
    CORRETTO BOOLEAN DEFAULT FALSE,  
    VOTO_TEST FLOAT DEFAULT 0,  
    PRIMARY KEY (USERNAME_I, USERNAME_S, ID_TEST),  
    FOREIGN KEY (USERNAME_I) REFERENCES INSEGNANTE (USERNAME_I),  
    FOREIGN KEY (USERNAME_S) REFERENCES STUDENTE (USERNAME_S),  
    FOREIGN KEY (ID_TEST) REFERENCES TEST (ID_TEST)  
);
```

### 3.6.5 Quesito Aperto

```
CREATE TABLE QUESITO_APERTO(  
    ID_A INT NOT NULL,  
    ID_TEST INT NOT NULL,  
    PUNTEGGIO_MIN FLOAT NOT NULL,  
    PUNTEGGIO_MAX FLOAT NOT NULL,  
    DOMANDA VARCHAR(150) NOT NULL,  
    PRIMARY KEY (ID_A),  
    FOREIGN KEY (ID_TEST) REFERENCES TEST (ID_TEST)  
);
```

### 3.6.6 Risposta Aperta

```
CREATE TABLE RISPOSTA_APERTA(  
    IDR_A INT NOT NULL,  
    USERNAME_S VARCHAR(25) NOT NULL,  
    USERNAME_I VARCHAR(25) NOT NULL,  
    RISPOSTA_DATA VARCHAR(500),  
    COMMENTO VARCHAR(500) DEFAULT 'Nessun commento.',  
    PUNTEGGIO_RISA FLOAT DEFAULT 0,  
    PRIMARY KEY (IDR_A, USERNAME_S),  
    FOREIGN KEY (USERNAME_S) REFERENCES STUDENTE (USERNAME_S),  
    FOREIGN KEY (USERNAME_I) REFERENCES INSEGNANTE (USERNAME_I),  
    FOREIGN KEY (IDR_A) REFERENCES QUIZ_APERTO (ID_A)  
);
```

### 3.6.7 Quesito Multiplo

```
CREATE TABLE QUESITO_MULTIPLO(  
    ID_M INT NOT NULL,  
    ID_TEST INT NOT NULL,  
    PUNTEGGIO_CORRETTO FLOAT NOT NULL,  
    PUNTEGGIO_ERRATO FLOAT NOT NULL,  
    DOMANDA VARCHAR(200) NOT NULL,  
    R_UNOC VARCHAR(150) NOT NULL,  
    R_DUE VARCHAR(150) NOT NULL,  
    R_TRE VARCHAR(150),  
    R_QUATTRO VARCHAR(50),  
    R_CINQUE VARCHAR(50),  
    PRIMARY KEY (ID_M),  
    FOREIGN KEY (ID_TEST) REFERENCES TEST (ID_TEST)  
);
```

### 3.6.8 Risposte Multiple

```
CREATE TABLE RISPOSTE_MULTIPLE(  
    IDR_M INT NOT NULL,  
    USERNAME_S VARCHAR(25) NOT NULL,  
    RISPOSTA_DATA VARCHAR(150) NOT NULL,  
    CORRETTA BOOLEAN,  
    PUNTEGGIO_RM FLOAT DEFAULT 0,  
    PRIMARY KEY (IDR_M, USERNAME_S),  
    FOREIGN KEY (IDR_M) REFERENCES QUIZ_MULTIPLO (ID_M),  
    FOREIGN KEY (USERNAME_S) REFERENCES STUDENTE (USERNAME_S)  
);
```



## 3.7 Implementazione vincoli sugli attributi di un'ennupla

### 3.7.1 Controllo Punteggio Aperto

```
ALTER TABLE QUIZ_APERTO
ADD CONSTRAINT controllo_pa CHECK (PUNTEGGIO_MIN<PUNTEGGIO_MAX);
```

### 3.7.2 Controllo Punteggio Multiplo

```
ALTER TABLE QUIZ_MULTIPLO
ADD CONSTRAINT controllo_pm1 CHECK (PUNTEGGIO_CORRETTO>PUNTEGGIO_ERRATO),
ADD CONSTRAINT controllo_pm2 CHECK (PUNTEGGIO_CORRETTO>0),
ADD CONSTRAINT controllo_pm3 CHECK (PUNTEGGIO_ERRATO<=0)
```

### 3.7.3 Controllo Tempo

```
CREATE OR REPLACE FUNCTION controllo_tempo() RETURNS TRIGGER AS $controllo_tempo$
DECLARE
tempo TIME;
BEGIN
SELECT TEST.TEMPO_SVOLGIMENTO INTO tempo
FROM TEST
WHERE TEST.TEMPO_SVOLGIMENTO = NEW.TEMPO_SVOLGIMENTO;

    IF(tempo < '00:15:00' AND tempo IS NOT NULL) THEN
RAISE EXCEPTION 'Tempo insufficiente per svolgere il test.'
USING HINT = 'Prova ad assegnare maggior tempo (>= 15 min)';
END IF;
RETURN NULL;
END;

$controllo_tempo$ LANGUAGE PLPGSQL;

CREATE TRIGGER CONTROLLO_TEMPO AFTER INSERT OR UPDATE OF TEMPO_SVOLGIMENTO
ON TEST
FOR EACH ROW EXECUTE PROCEDURE controllo_tempo();
```

### 3.7.4 Controllo credenziali utente

```
CREATE OR REPLACE FUNCTION controllo_lunghezza_stu() RETURNS TRIGGER AS $controllo_lunghezza_stu$
DECLARE
username STUDENTE.USERNAME_S%TYPE;
NOME STUDENTE.NOME%TYPE;
COGNOME STUDENTE.COGNOME%TYPE;
BEGIN

SELECT STUDENTE.USERNAME_S, STUDENTE.NOME, STUDENTE.COGNOME INTO username
FROM STUDENTE
WHERE STUDENTE.USERNAME_S = NEW.USERNAME_S AND STUDENTE.NOME = NEW.NOME AND STUDENTE.COGNOME = NEW.COGNOME;

IF(LENGTH(username) < 8) THEN
RAISE EXCEPTION 'Username non valido'
USING HINT = 'Prova a inserire un username più lungo';
END IF;
IF(LENGTH(nome) < 2 OR LENGTH(cognome) < 2) THEN
RAISE EXCEPTION 'Nome o cognome non valido'
USING HINT = 'Prova a inserire un nome e cognome vero!';
END IF;
RETURN NULL;
END;
```

```
$controllo_lunghezza_stu$ LANGUAGE PLPGSQL;
```

```
CREATE TRIGGER CONTROLLO_CREDENZIALI_STUDENTE AFTER INSERT  
ON STUDENTE  
FOR EACH ROW EXECUTE PROCEDURE controllo_lunghezza_stu();
```

```
CREATE OR REPLACE FUNCTION controllo_lunghezza_ins() RETURNS TRIGGER AS $controllo_lunghezza_ins$  
DECLARE  
username INSEGNANTE.USERNAME_I%TYPE;  
password_i INSEGNANTE.PASSWORD_I%TYPE;  
nome INSEGNANTE.NOME%TYPE;  
cognome INSEGNANTE.COGNOME%TYPE;  
BEGIN  
  
SELECT INSEGNANTE.USERNAME_I, INSEGNANTE.PASSWORD_I, INSEGNANTE.NOME, INSEGNANTE.COGNOME INTO  
username, password_i, nome, cognome  
FROM INSEGNANTE  
WHERE INSEGNANTE.USERNAME_I = NEW.USERNAME_I AND INSEGNANTE.PASSWORD_I = NEW.PASSWORD_I AND  
INSEGNANTE.NOME = NEW.NOME AND INSEGNANTE.COGNOME = NEW.COGNOME;  
  
IF(LENGTH(username) < 8 OR LENGTH(password_i) < 8) THEN  
RAISE EXCEPTION 'Username non valido'  
USING HINT = 'Prova a inserire un username più lungo';  
END IF;  
IF(LENGTH(nome) < 2 OR LENGTH(cognome) < 2) THEN  
RAISE EXCEPTION 'Nome o cognome non valido'  
USING HINT = 'Prova a inserire un nome e cognome vero!';  
END IF;  
RETURN NULL;  
END;  
  
$controllo_lunghezza_ins$ LANGUAGE PLPGSQL;
```

```
CREATE TRIGGER CONTROLLO_CREDENZIALI_INSEGNANTE AFTER INSERT  
ON INSEGNANTE  
FOR EACH ROW EXECUTE PROCEDURE controllo_lunghezza_ins();
```

### 3.7.5 Controllo password utente

```
CREATE OR REPLACE FUNCTION verifica_ipassword() RETURNS TRIGGER AS $verifica_ipassword$  
DECLARE  
pass insegnante.password_i%type;  
BEGIN  
SELECT INSEGNANTE.PASSWORD_I INTO pass  
FROM INSEGNANTE  
WHERE INSEGNANTE.PASSWORD_I = NEW.PASSWORD_I;  
  
IF(LENGTH(pass) < 8) THEN  
RAISE EXCEPTION 'Password % non valida', pass  
USING HINT = 'Prova a inserire una password più lunga!';  
END IF;  
RETURN NULL;  
END;  
  
$verifica_ipassword$ LANGUAGE PLPGSQL;
```

```

CREATE TRIGGER VERIFICA_IPASSWORD AFTER INSERT
ON INSEGNANTE
FOR EACH ROW EXECUTE PROCEDURE verifica_ipassword();

CREATE OR REPLACE FUNCTION verifica_spasword() RETURNS TRIGGER AS $verifica_spasword$
DECLARE
spasword STUDENTE.PASSWORD_S%TYPE;
BEGIN
SELECT PASSWORD_S INTO spasword
FROM STUDENTE
WHERE STUDENTE.PASSWORD_S = NEW.PASSWORD_S;

IF(LENGTH(spasword) < 8) THEN
RAISE EXCEPTION 'Password non valida'
USING HINT = 'Prova a inserire una password più lunga!';
END IF;
RETURN NULL;
END;

$verifica_spasword$ LANGUAGE PLPGSQL;

CREATE TRIGGER VERIFICA_SPASSWORD AFTER INSERT
ON STUDENTE
FOR EACH ROW EXECUTE PROCEDURE verifica_spasword();

```

## 3.8 Implementazione vincoli Interrelazionali

### 3.8.1 Controllo punteggio Assegnato

```
CREATE OR REPLACE FUNCTION controllo_risa ()
RETURNS TRIGGER AS $controllo_risa$
DECLARE
Var1 RISPOSTA_APERTA.PUNTEGGIO_RISA%type;
Var2 QUESITO_APERTO.PUNTEGGIO_MAX%type;
Var3 QUESITO_APERTO.PUNTEGGIO_MIN%type;
idquiz RISPOSTA_APERTA.IDR_A%TYPE;
BEGIN
SELECT RISPOSTA_APERTA.PUNTEGGIO_RISA, RISPOSTA_APERTA.IDR_A INTO VAR1, idquiz
FROM RISPOSTA_APERTA
WHERE PUNTEGGIO_RISA = NEW.PUNTEGGIO_RISA;

SELECT QUESITO_APERTO.PUNTEGGIO_MAX, QUESITO_APERTO.PUNTEGGIO_MIN INTO VAR2, VAR2
FROM QUIZ_APERTO
WHERE QUESITO_APERTO.ID_A = idquiz;

IF (VAR1>VAR2 AND VAR1<VAR3)
THEN
RAISE EXCEPTION 'ERROR punteggio non valido';
RETURN NULL;
END IF;
END;
$controllo_risa$
LANGUAGE plpgsql;

CREATE TRIGGER CONTROLLO_RISA AFTER UPDATE OF PUNTEGGIO_RISA
ON RISPOSTA_APERTA
FOR EACH ROW EXECUTE PROCEDURE controllo_risa();
```

### 3.8.2 Controllo Valore booleano

```
CREATE OR REPLACE FUNCTION assegna_bool() RETURNS TRIGGER AS $assegna_bool$
DECLARE
Rispostad RISPOSTE_APERTE.RISPOSTA_DATA%type;
runo QUESITO_MULTIPLIO.R_UNOC%type;
idrm RISPOSTE_MULTIPLE.IDR_M%type;
piu QUESITO_MULTIPLIO.PUNTEGGIO_ESATTO%TYPE;
meno QUESITO_MULTIPLIO.PUNTEGGIO_ERRATO%TYPE;

BEGIN

SELECT RISPOSTA_APERTA.RISPOSTA_DATA INTO rispostad, idrm
FROM RISPOSTE_APERTE
WHERE RISPOSTA_APERTA.RISPOSTA_DATA = NEW.RISPOSTA_DATA;

SELECT QUESITO_MULTIPLIO.R_UNOC, QUESITO_MULTIPLIO.PUNTEGGIO_ERRATO, QUESITO_MULTIPLIO.PUNTEGGIO_ESATTO INTO runo
FROM QUESITO_MULTIPLIO
WHERE QUESITO_MULTIPLIO.ID_M = idrm;

IF(rispostad = runo) THEN
UPDATE RISPOSTE_MULTIPLE
SET NEW.CORRETTA = TRUE
WHERE RISPOSTE_MULTIPLE.IDR_M = idrm;

UPDATE RISPOSTE_MULTIPLE
SET NEW.PUNTEGGIO_RISM = piu
WHERE RISPOSTE_MULTIPLE.IDR_M = idrm;
```

```

END IF;
IF (rispostad <> runo) THEN
UPDATE RISPOSTE_MULTIPLE
SET NEW.CORRETTA = FALSE
WHERE RISPOSTE_MULTIPLE.IDR_M = idrm;

```

```

UPDATE RISPOSTE_MULTIPLE
SET NEW.PUNTEGGIO_RISM = meno
WHERE RISPOSTE_MULTIPLE.IDR_M = idrm;
END IF;
RETURN NULL;
END;

```

```

$assegna_bool$ LANGUAGE PLPGSQL;

```

```

CREATE TRIGGER ASSEGNA_BOOL AFTER INSERT OR UPDATE OF RISPOSTA_DATA
ON RISPOSTE_MULTIPLE
FOR EACH ROW EXECUTE PROCEDURE assegna_bool();

```

### 3.8.3 Controllo Dominio risposta Multipla

```

CREATE OR REPLACE FUNCTION controllo_risposta() RETURNS TRIGGER AS $controllo_risposta$
DECLARE
rispostaData RISPOSTE_MULTIPLE.RISPOSTA_DATA%type;
rispostaU RISPOSTE_MULTIPLE.RISPOSTA_DATA%type;
rispostaD RISPOSTE_MULTIPLE.RISPOSTA_DATA%type;
rispostaT RISPOSTE_MULTIPLE.RISPOSTA_DATA%type;
rispostaQ RISPOSTE_MULTIPLE.RISPOSTA_DATA%type;
rispostaC RISPOSTE_MULTIPLE.RISPOSTA_DATA%type;

BEGIN
SELECT RISPOSTE_MULTIPLE.RISPOSTA_DATA INTO rispostaData
FROM RISPOSTE_MULTIPLE
WHERE RISPOSTE_MULTIPLE.RISPOSTA_DATA = NEW.RISPOSTADATA;

SELECT QUESITO_MULTIPLO.R_UNOC INTO rispostaU
FROM QUESITO_MULTIPLO
WHERE QUESITO_MULTIPLO.ID_M = RISPOSTE_MULTIPLE.IDR_M;

SELECT QUESITO_MULTIPLO.R_DUE INTO rispostaD
FROM QUESITO_MULTIPLO
WHERE QUESITO_MULTIPLO.ID_M = RISPOSTE_MULTIPLE.IDR_M;

SELECT QUESITO_MULTIPLO.R_TRE INTO rispostaT
FROM QUESITO_MULTIPLO
WHERE QUESITO_MULTIPLO.ID_M = RISPOSTE_MULTIPLE.IDR_M;

SELECT QUESITO_MULTIPLO.R_QUATTRO INTO rispostaQ
FROM QUESITO_MULTIPLO
WHERE QUESITO_MULTIPLO.ID_M = RISPOSTE_MULTIPLE.IDR_M;

SELECT QUESITO_MULTIPLO.R_CINQUE INTO rispostaC
FROM QUESITO_MULTIPLO
WHERE QUESITO_MULTIPLO.ID_M = RISPOSTE_MULTIPLE.IDR_M;

```

```

IF(rispostaData <> rispostaU AND rispostaData<> rispostaD
AND rispostaT IS NULL AND rispostaQ IS NULL AND rispostaC IS NULL)
THEN RAISE EXCEPTION 'Risposta data non valida.';
END IF;

IF(rispostaData <> rispostaU AND rispostaData<> rispostaD
AND (rispostaT IS NOT NULL AND rispostaData <> rispostaT) AND rispostaQ IS NULL AND rispostaC IS NULL)
THEN RAISE EXCEPTION 'Risposta data non valida.';
END IF;

IF(rispostaData <> rispostaU AND rispostaData<> rispostaD
AND (rispostaT IS NOT NULL AND rispostaData <> rispostaT) AND
(rispostaQ IS NOT NULL AND rispostaD<> rispostaQ) AND rispostaC IS NULL)
THEN RAISE EXCEPTION 'Risposta data non valida.';
END IF;

IF (rispostaData <> rispostaU AND rispostaData<> rispostaD
AND (rispostaT IS NOT NULL AND rispostaData <> rispostaT) AND
(rispostaQ IS NOT NULL AND rispostaD<> rispostaQ) AND
(rispostaC IS NOT NULL AND rispostaD <> rispostaC))
THEN RAISE EXCEPTION 'Risposta data non valida.';
END IF;

RETURN NULL;
END;

$controllo_risposta$ LANGUAGE PLPGSQL;

CREATE TRIGGER CONTROLLO_RISPOSTA_VALIDA AFTER INSERT OR UPDATE OF RISPOSTA_DATA
ON RISPOSTE_MULTIPLE
FOR EACH ROW EXECUTE PROCEDURE controllo_risposta();

```

### 3.8.4 Controllo username

```

CREATE OR REPLACE FUNCTION controllo_username_i() RETURNS TRIGGER AS $controllo_username_i$
DECLARE
username_s VARCHAR(25);
username_i VARCHAR(25);
BEGIN
SELECT INSEGNANTE.USERNAME_I INTO username_i
FROM INSEGNANTE
WHERE INSEGNANTE.USERNAME_I = NEW.USERNAME_I;

SELECT STUDENTE.USERNAME_S INTO username_s
FROM STUDENTE
WHERE STUDENTE.USERNAME_S = username_i;

IF(username_s = username_i AND username_s IS NOT NULL) THEN
RAISE EXCEPTION 'Questo username non è disponibile.'
USING HINT = 'Prova con un inserire un username diverso.';
END IF;
RETURN NULL;
END;

$controllo_username_i$ LANGUAGE PLPGSQL;

CREATE TRIGGER CONTROLLO_USERNAME_I AFTER INSERT OR UPDATE OF USERNAME_I
ON INSEGNANTE

```

```

FOR EACH ROW EXECUTE PROCEDURE controllo_username_i();

CREATE OR REPLACE FUNCTION controllo_username_s() RETURNS TRIGGER AS $controllo_username_s$
DECLARE
username_s VARCHAR(25);
username_i VARCHAR(25);
BEGIN

    SELECT STUDENTE.USERNAME_S INTO username_s
FROM STUDENTE
WHERE STUDENTE.USERNAME_S = NEW.USERNAME_S;

SELECT INSEGNANTE.USERNAME_I INTO username_i
FROM INSEGNANTE
WHERE INSEGNANTE.USERNAME_I = username_s;

IF(username_s = username_i AND username_i IS NOT NULL) THEN
RAISE EXCEPTION 'Questo username non è disponibile.'
USING HINT = 'Prova con un inserire un username diverso.';
END IF;
RETURN NULL;
END;

$controllo_username_s$ LANGUAGE PLPGSQL;

CREATE TRIGGER CONTROLLO_USERNAME_S AFTER INSERT OR UPDATE OF USERNAME_S
ON STUDENTE
FOR EACH ROW EXECUTE PROCEDURE controllo_username_s();

```

### 3.8.5 Controllo creatore del test

```

CREATE OR REPLACE FUNCTION verifica_insegnanteCorrezione() RETURNS TRIGGER AS
$verifica_insegnanteCorrezione$
BEGIN

IF NOT EXISTS (SELECT TEST.USERNAME_I FROM TEST, CORREZIONE WHERE TEST.USERNAME_I = NEW.USERNAME_I)
RAISE EXCEPTION 'L'insegnante % non ha creato nessun test', new.username;
END IF;

RETURN NULL;
END;
$verifica_insegnanteCorrezione$ LANGUAGE PLPGSQL;

CREATE TRIGGER verifica_insegnanteCorrezione AFTER INSERT ON CORREZIONE
FOR EACH ROW EXECUTE PROCEDURE verifica_insegnanteCorrezione();

```

## 3.9 Implementazione Automazioni

### 3.9.1 Aggiornamento voto

```

CREATE OR REPLACE FUNCTION aggiorna_voto_a() RETURNS TRIGGER AS $aggiorna_voto_a$
DECLARE
punteggio RISPOSTA_APERTA.PUNTEGGIO_RISA%TYPE;
username RISPOSTA_APERTA.USERNAME_S%TYPE;
idra RISPOSTA_APERTA.IDR_A%TYPE;

```

```

idTestQ QUESITO_APERTO.ID_TEST%TYPE;
voto CORREZIONE.VOTO_TEST%TYPE;

BEGIN
SELECT RISPOSTA_APERTA.PUNTEGGIO_RISA, RISPOSTA_APERTE.USERNAME_S, RISPOSTA_APERTA.IDR_A INTO punteggio, usern
FROM RISPOSTA_APERTA
WHERE RISPOSTA_APERTA.PUNTEGGIO_RISA = NEW.PUNTEGGIO_RISA;

SELECT QUESITO_APERTO.ID_TEST INTO idTestQ
FROM QUESITO_APERTO
WHERE QUESITO_APERTO.ID_A = idra;

SELECT CORREZIONE.VOTO_TEST INTO voto
FROM CORREZIONE
WHERE CORREZIONE.ID_TEST = idTestQ;

voto := voto + punteggio;

UPDATE CORREZIONE
SET CORREZIONE.VOTO_TEST = voto
WHERE CORREZIONE.USERNAME_S = username;

RETURN NULL;
END;

$aggiorna_voto_a$ LANGUAGE PLPGSQL;

CREATE TRIGGER AGGIORNA_VOTO_A AFTER UPDATE OF PUNTEGGIO_RISA
ON RISPOSTA_APERTA
FOR EACH ROW EXECUTE PROCEDURE aggiorna_voto_a();

CREATE OR REPLACE FUNCTION somma_mul() RETURNS TRIGGER AS $somma_mul$
DECLARE
punteggio RISPOSTE_MULTIPLE.PUNTEGGIO_RM%TYPE;
username RISPOSTE_MULTIPLE.USERNAME_S%TYPE;
idrm RISPOSTE_MULTIPLE.IDR_M%TYPE;
idTestQ QUESITO_MULTIPLO.ID_TEST%TYPE;
voto CORREZIONE.VOTO_TEST%TYPE;
idTestC CORREZIONE.ID_TEST%TYPE;

BEGIN
SELECT RISPOSTE_MULTIPLE.PUNTEGGIO_RM, RISPOSTE_MULTIPLE.USERNAME_S, RISPOSTE_MULTIPLE.IDR_M INTO
punteggio, username, idrm
FROM RISPOSTE_MULTIPLE
WHERE RISPOSTE_MULTIPLE.PUNTEGGIO_RM = NEW.PUNTEGGIO_RM;

SELECT QUESITO_MULTIPLO.ID_TEST INTO idTestQ
FROM QUESITO_MULTIPLO
WHERE QUESITO_MULTIPLO.ID_M = idrm;

SELECT CORREZIONE.VOTO_TEST INTO voto
FROM CORREZIONE
WHERE CORREZIONE.ID_TEST = idTestQ;

voto := voto + punteggio;

```



```

UPDATE CORREZIONE
SET CORREZIONE.VOTO_TEST = voto
WHERE CORREZIONE.USERNAME_S = username;

RETURN NULL;
END;

$somma_mul$ LANGUAGE PLPGSQL;

CREATE TRIGGER SOMMA_PUNTEGGIOM AFTER UPDATE OF PUNTEGGIO_RM
ON RISPOSTE_MULTIPLE
FOR EACH ROW EXECUTE PROCEDURE somma_mul();

```

### 3.9.2 Aggiornamento punteggio totale

```

CREATE OR REPLACE FUNCTION aggiorna_punteggiotot() RETURNS TRIGGER AS $aggiorna_punteggiotot$
DECLARE
username STUDENTE.USERNAME_S%TYPE;
punteggio STUDENTE.PUNTEGGIO_TOT%TYPE;
Voto CORREZIONE.VOTO_TEST%TYPE;
BEGIN

SELECT CORREZIONE.VOTO_TEST, CORREZIONE.USERNAME_S INTO voto, username
FROM CORREZIONE
WHERE CORREZIONE.VOTO_TEST = NEW.VOTO_TEST;

    SELECT STUDENTE.PUNTEGGIO_TOT INTO punteggio
FROM STUDENTE
WHERE STUDENTE.USERNAME_S = username;

punteggio := punteggio + voto;
    UPDATE STUDENTE
SET STUDENTE.PUNTEGGIO_TOT = punteggio
WHERE (STUDENTE.USERNAME_S = username);
RETURN NULL;
END;

$aggiorna_punteggiotot$ LANGUAGE PLPGSQL;

CREATE TRIGGER AGGIORNA_PUNTEGGIOTOT AFTER UPDATE OF VOTO_TEST
ON CORREZIONE
FOR EACH ROW EXECUTE PROCEDURE aggiorna_punteggiotot();

```

## 3.10 Implementazioni viste

### 3.10.1 Visualizza dati studente

```

CREATE OR REPLACE VIEW visualizza_dati_studente (NOME_TEST, N_STUDENTI)
AS (SELECT NOME_TEST, COUNT(USERNAME_S)
    FROM CORREZIONE AS C, TEST AS T
    WHERE C.ID_TEST = T.ID_TEST
    GROUP BY NOME_TEST)

```

### 3.10.2 Risposte esatte per studente

```

CREATE OR REPLACE VIEW risposte_esatte (STUDENTE, ESATTE, TEST)
AS (SELECT R.USERNAME_S, COUNT(R.PUNTEGGIO_RISA) + COUNT(W.CORRETTA), Q.ID_TEST

```

```

FROM RISPOSTA_APERTA AS R, QUESITO_APERTO AS Q, QUESITO_MULTIPLO AS M,
RISPOSTE_MULTIPLE AS W, RISPOSTA_APERTA AS F, RISPOSTE_MULTIPLE AS Y
WHERE (R.PUNTEGGIO_RISA>PUNTEGGIO_MIN OR W.CORRETTA=TRUE) AND (R.USERNAME_S = W.USERNAME_S) AND
(R.USERNAME_S = F.USERNAME_S) AND (W.USERNAME_S = Y.USERNAME_S) AND (Q.ID_TEST = M.ID_TEST) AND
(M.ID_M = W.IDR_M) AND (R.IDR_A = Q.ID_A)
GROUP BY R.USERNAME_S, Q.ID_TEST)

```

### 3.10.3 Risposte errate per studente

```

CREATE OR REPLACE VIEW risposte_errate (STUDENTE, ERRATE, TEST)
AS (SELECT R.USERNAME_S, COUNT(R.PUNTEGGIO_RISA) + COUNT(W.CORRETTA), Q.ID_TEST
FROM RISPOSTA_APERTA AS R, QUESITO_APERTO AS Q, QUESITO_MULTIPLO AS M,
RISPOSTE_MULTIPLE AS W, RISPOSTA_APERTA AS F, RISPOSTE_MULTIPLE AS Y
WHERE (R.PUNTEGGIO_RISA = PUNTEGGIO_MIN OR W.CORRETTA=FALSE) AND
(R.USERNAME_S = W.USERNAME_S) AND (R.USERNAME_S = F.USERNAME_S)
AND (W.USERNAME_S = Y.USERNAME_S) AND (Q.ID_TEST = M.ID_TEST) AND (M.ID_M = W.IDR_M) AND (R.IDR_A = Q.ID_A)
GROUP BY R.USERNAME_S, Q.ID_TEST)

```

### 3.10.4 Visualizza risultati test

```

CREATE OR REPLACE VIEW visualizza_risultati (NOME_STU, NOME_TEST, NOME_INS, RIS_ESATTE, RIS_ERRATE, VOTO)
AS (SELECT C.USERNAME_S, T.NOME_TEST, C.USERNAME_I, E.ESATTE, R.ERRATE, C.VOTO_TEST,
FROM CORREZIONE AS C, RISPOSTE_ESATTE AS E, RISPOSTE_ERRATE AS R, TEST AS T, TEST AS Y
WHERE T.ID_TEST = Y.ID_TEST)

```

### 3.10.5 Visualizza media per materia

```

CREATE OR REPLACE VIEW media_categoria (NOME_TEST, MATERIA, STUDENTE, MEDIA) AS
(SELECT T.NOME_TEST, T.MATERIA_TEST, C.USERNAME_S, AVG(C.VOTO_TEST)
FROM TEST AS T, TEST AS P, CORREZIONE AS C
GROUP BY T.NOME_TEST, T.MATERIA_TEST, C.USERNAME_S, P.MATERIA_TEST
HAVING T.MATERIA_TEST = P.MATERIA_TEST
)

```

Per il file SQL e la popolazione, vedasi il seguente link.

Giorgio Longobardo  
Domitilla Giulia Simeoli