

# Relazione progetto di Basi di Dati

SIMONE SAMOGGIA 970758

## 1. Raccolta e analisi dei requisiti

### 1.1 Specifiche

La piattaforma ESQL si basa sul database relazionale ESQldb. Tutti gli utenti della piattaforma dispongono di: email, nome, cognome, eventuale recapito telefonico. Gli utenti sono divisi in due tipologie: docenti e studenti. I primi dispongono anche di: nome del dipartimento di appartenenza e nome del corso di cui sono titolari. I secondi dispongono anche di un campo anno di immatricolazione e di un codice alfanumerico di lunghezza pari a 16 caratteri. I docenti possono creare delle tabelle SQL (definite tabelle di esercizio, nel seguito): ogni tabella di esercizio dispone di nome, data di creazione, un campo num\_righe. Inoltre, ogni tabella di esercizio dispone di un insieme di attributi: ogni attributo dispone di un nome, un tipo, e può far parte della chiave primaria della tabella di esercizio. Devono poter essere inseriti dai docenti anche gli vincoli di integrità tra attributi di diverse tabelle di esercizio. In aggiunta, ogni docente può creare dei test: ogni test dispone di un titolo univoco, una data di creazione ed un'eventuale foto. Ogni test può includere una serie di quesiti: ogni quesito dispone di un numero progressivo (univoco, ma solo all'interno di uno specifico test), un livello di difficoltà (campo enum con valori: Basso, Medio, Alto), un campo descrizione, un campo `#numrisposte` (ridondanza concettuale, vedere specifiche sotto) e fa riferimento ad una o più tabelle di esercizio tra quelle create dal docente. I quesiti possono appartenere esclusivamente a due categorie: quesiti a risposta chiusa o quesiti di codice. Nel primo caso, il quesito dispone di una serie di opzioni di risposta: ogni opzione dispone di una numerazione (univoca, ma solo all'interno di uno specifico quesito) ed un campo testo. Nel secondo caso, il quesito dispone di una o più soluzioni (sketch di codice SQL che implementano correttamente quanto richiesto dalla descrizione del quesito). Ogni test dispone di un campo booleano VisualizzaRisposte: se settato a True, le risposte dei quesiti diventano visibili agli studenti, altrimenti restano nascoste. Gli studenti possono svolgere un test, inserendo una o più risposte per ciascun quesito. Si vuole tenere traccia del completamento del test, ossia: data di inserimento della prima risposta (su scala temporale), data di inserimento dell'ultima risposta (su scala temporale), stato (campo enum con tre valori: Aperto, InCompletamento, Concluso). Nel caso di quesiti a risposta chiusa, la risposta consiste nell'opzione scelta tra quelle disponibili. Nel caso di quesiti di codice, la risposta consiste in un campo testo (codice SQL che risolve l'esercizio). E' prevista la possibilità per lo studente di sottomettere più risposte per lo stesso quesito, in istanti diversi di tempo, ma solo se il test non è

stato Concluso. Ogni risposta dispone di un campo esito, che può valere True o False a seconda che la risposta fornita dallo studente coincida con quella inserita dal docente (nel caso di quesiti a risposta chiusa) o che la risposta fornita dallo studente produca lo stesso output di quella inserita dal docente (nel caso di quesiti di codice). Infine, è prevista la possibilità di inviare messaggi nella piattaforma. Ogni messaggio dispone di un titolo, un campo testo, una data di inserimento, e fa riferimento ad uno specifico test. Il messaggio può essere inviato da un docente: in tal caso, il messaggio viene ricevuto da tutti gli studenti. Viceversa, un messaggio può essere inviato da uno studente: in tal caso, il destinatario è uno specifico docente.

Infine, si vuole tenere traccia di tutti gli eventi che occorrono nella piattaforma, relativamente all'inserimento di nuovi dati (es. nuovi utenti, nuovi test, nuovi quesiti, etc). Tali eventi vanno inseriti, sotto forma di messaggi di testo, all'interno di un log, implementato in un' apposita collezione MongoDB.

## **1.2 Lista delle operazioni**

### **Operazioni di tutti gli utenti**

- Autenticazione/registrazione sulla piattaforma
- Visualizzazione dei test disponibili
- Visualizzazione dei quesiti presenti all'interno di ciascun test

### **Operazioni esclusive dei docenti**

- Inserimento di una nuova tabella di esercizio, con relativi meta-dati
- Inserimento di una riga per una tabella di esercizio definita dal docente.
- Creazione di nuovo test
- Creazione di un nuovo quesito con le relative risposte
- Abilitare / disabilitare la visualizzazione delle risposte per uno specifico test
- Inserimento di un messaggio (broadcast)

### **Operazioni esclusive degli studenti**

- Inserimento di una nuova risposta ad un quesito
- Visualizzazione dell'esito della risposta
- Inserimento di un messaggio

## **Statistiche visualizzabili da tutti gli utenti**

- Visualizzare la classifica degli studenti, sulla base del numero di test completati (un test si considera completato se il suo stato è pari a "Concluso"). Nella classifica NON devono apparire i dati sensibili dello studente (nome, cognome, email) ma solo il codice alfanumerico.
- Visualizzare la classifica degli studenti, sulla base del numero di risposte corrette inserite rispetto al numero totale di risposte inserite. Nella classifica NON devono apparire i dati sensibili dello studente (nome, cognome, email) ma solo il codice alfanumerico.
- Visualizzare la classifica dei quesiti, in base al numero di risposte inserite dagli studenti.

## **1.3 Tavola dei volumi**

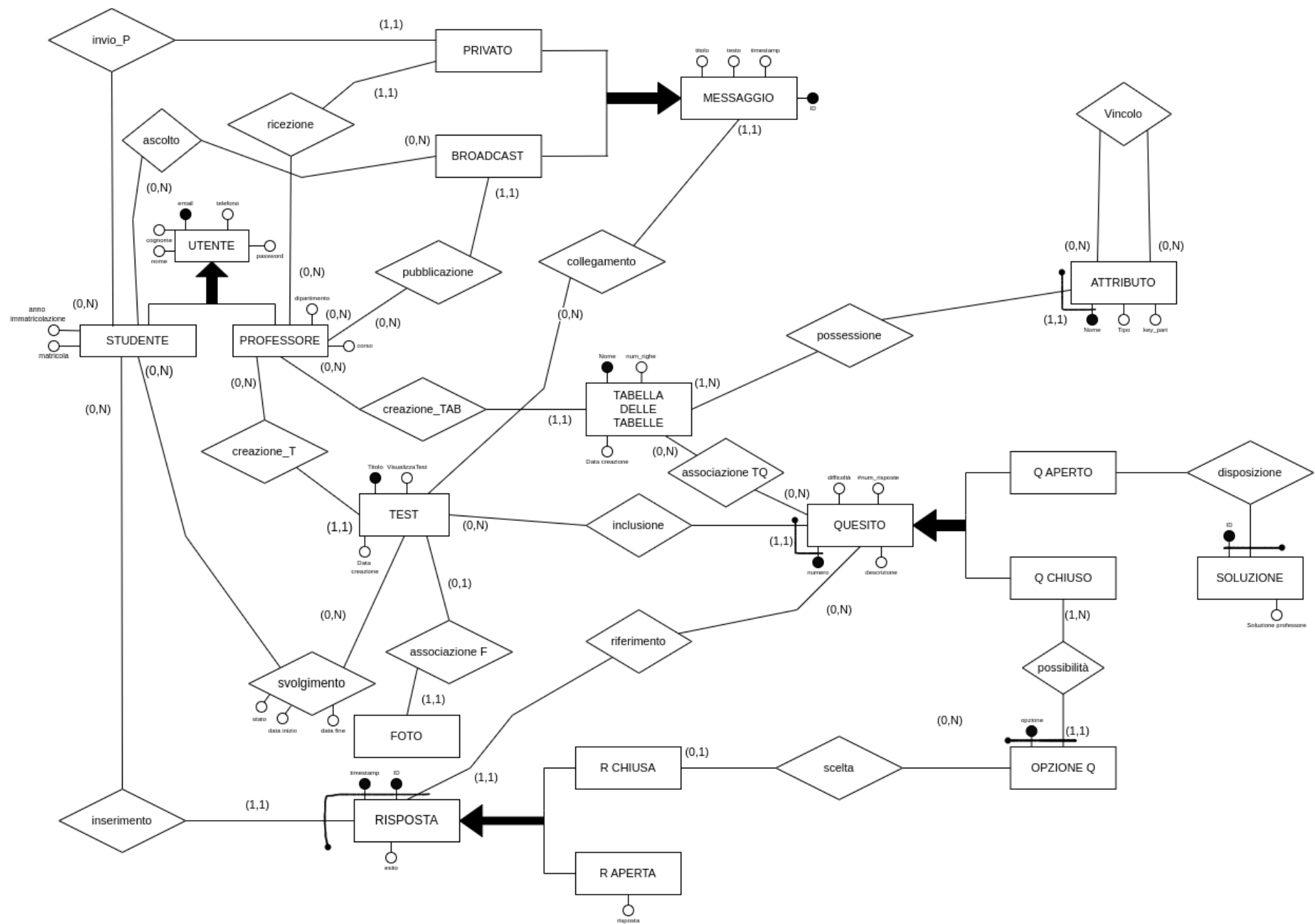
- 10 risposte per quesito
- 20 quesiti
- 50 utenti

## **1.4 Glossario**

Non ci sono termini specifici utilizzati all'interno della piattaforma

# **2. Progettazione concettuale**

## **2.1 Diagramma ER**



## **2.2 Dizionario delle entità**

Entità	Descrizione	Attributi	Indentificatore
BROADCAST	Messaggio inviato dal professore a tutti gli studenti	Id_messaggio, mittente	id_messaggio
FOTO_TEST	Contiene le foto dei test	foto, test_associato	test_associato
MESSAGGIO	Contiene le informazioni su un messaggio	ID, titolo, testo, data_inserimento, test_associato	ID
MESSAGGIO_PRIVATO	Messaggio inviato da uno studente ad un professore	id_messaggio, mittente, destinatario	id_messaggio
PROFESSORE	Contiene tutti i professori	email_professore, dipartimento, corso	email_professore
QUESITO	Tabella che contiene le informazioni generali e comuni di un quesito	ID, numero_quesito, test_associato, descrizione, livello_difficoltà, numero_risposte, tipo_quesito	ID
QUESITO_APERTO_SOLUZIONE	Soluzioni di un quesito aperto	id_quesito, id_soluzione, soluzione_professore	id_soluzione
QUESITO_CHIUSO_OPZIONE	Opzioni quesito chiuso	numero_opzione, id_quesito, testo, is_corretta	numero_opzione, id_quesito
RISPOSTA	Risposta inserita da uno studente	ID, id_quesito, email_studente, TIMESTAMP, tipo_risposta, esito	ID
RISPOSTA_QUESITO_APERTO	Query inserita dallo studente per la risposta	id_risposta, risposta	id_risposta
RISPOSTA_QUESITO_CHIUSO	Opzione scelta dallo studente per la risposta	id_risposta, opzione_scelta	id_risposta
STUDENTE	Utente studente	email_studente, matricola, anno_immatricolazione	email_professore
TABELLA DELLE TABELLE	Dizionario di tutte le tabelle inserite dai professori	nome_tabella, data_creazione, num_righe, creatore	nome_tabella
TAB_ATT	Contiene tutti gli attributi delle tabelle create dai professori	ID, nome_tabella, nome_attributo, tipo_attributo, key_part	ID
TEST	Test che crea il professore e che gli studenti devono svolgere	titolo, dataCreazione, VisualizzaRisposte, email_professore	titolo
UTENTE	Tutti gli utenti registrati nella piattaforma, contiene le informazioni generiche	email, nome, cognome, PASSWORD, tipo_utente, telefono	email

---

## **2.3 Dizionario delle relazioni**

RELAZIONE	Descrizione	Componenti	Attributi
VINCOLO	Collega un attributo di una tabella ad un attributo di un'altra tabella	TAB_ATT, TAB_ATT	
SVOLGIMENTO	Relazione che collega le risposte gli studenti ai test e contiene le informazioni sul ciclo di vita di esso	STUDENTE, TEST	data_inizio, data_fine, stato
ASSOCIAZIONE_QT	Relazione che collega uno specifico quesito di un test ad una tabella di esercizio	QUESITO, TABELLA DELLE TABELLE	
INVIO_P	collega uno studente al messaggio che ha inviato ad un professore	STUDENTE, PRIVATO	
RICEZIONE	collega un professore ad un messaggio privato	PROFESSORE, PRIVATO	
ASCOLTO	Collega uno studente ad un broadcast inviato dal professore	STUDENTE, BROADCAST	
PUBBLICAZIONE	Collega un professore al broadcast che ha creato	PROFESSORE, BROADCAST	
CREAZIONE_T	Collega il professore al test che ha creato	PROFESSORE, TEST	
CREAZIONE_TAB	Collega il professore alle tabelle di esercizio	PROFESSORE, TABELLA DELLE TABELLE	
COLLEGAMENTO	Relazione tra un messaggio e un test associato ad esso	MESSAGGIO, TEST	
POSSESSIONE	Collega il dizionario delle tabelle create ai loro attributi	TABELLA DELLE TABELLE, TAB_ATT	
INCLUSIONE	Relazione di appartenenza tra il test e i suoi quesiti	TEST, QUESITI	
DISPOSIZIONE	Collegamento tra le soluzioni inserite dal professore il quesito	Q_APERTO, SOLUZIONE	
POSSIBILITÀ	Collegamento tra le possibili opzioni e il relativo quesito chiuso	Q_CHIUSO, OPZIONE_Q	
INSERIMENTO	Relazione che mette in collegamento lo studente e la risposta inserita	STUDENTE, RISPOSTA	
RIFERIMENTO	Collega le risposte ai quesiti	RISPOSTA, QUESITO	
SCELTA	Collega la risposta chiusa data da uno studente e le opzioni di un quesito chiuso	R_CHIUSA, OPZIONE_Q	
ASSOCIAZIONE_F	Associa l'eventuale foto al relativo	TEST. FOTO TEST	



## 2.4 Tavola delle Business Rules

### REGOLE

- 1) Lo STUDENTE non può inserire una RISPOSTA dopo che il suo svolgimento TEST è stato reputato SVOLGIMENTO.stato => 'CONCLUSO'
- 2) Il TEST viene considerato 'CONCLUSO' da uno STUDENTE se tutte le sue RISPOSTA sono 'GIUSTA' oppure se il PROFESSORE decide di mostrare le RISPOSTA
- 3) RISPOSTA.esito è un ENUM ('GIUSTA', 'SBAGLIATA')
- 4) RISPOSTA.tipo\_risposta è un ENUM('APERTA', 'CHIUSA')
- 5) UTENTE.tipo\_utente è un ENUM('STUDENTE', 'PROFESSORE')
- 6) QUESITO.livello\_difficoltà è un ENUM('BASSO', 'MEDIO', 'ALTO')
- 7) QUESITO\_CHIUSO\_OPZIONE.is\_corretta è un ENUM('TRUE', 'FALSE')
- 8) TAB\_ATT.key\_part è un ENUM('TRUE', 'FALSE')
- 9) SVOLGIMENTO\_TEST.stato è un ENUM ('APERTO' , 'IN\_COMPLETAMENTO', 'CONCLUSO')
- 10) Un UTENTE dev'essere necessariamente o uno STUDENTE o un PROFESSORE
- 11) Un QUESITO dev'essere necessariamente APERTO o CHIUSO
- 12) Una RISPOSTA fa dev'essere dello stesso tipo del QUESITO
- 13) Lo STUDENTE può vedere le RISPOSTA solo se il PROFESSORE ha concluso il TEST
- 14) In TAB\_ATT dev'esserci almeno un valore TAB\_ATT.key\_part impostato come TRUE per ogni insieme TAB\_ATT.nome\_tabella
- 15) Una R\_APERTA per esser considerata giusta deve dare lo stesso output di una delle Q\_APERTO\_SOLUZIONE.soluzione\_professore ed è salvata come un TEXT
- 16) Una R\_CHIUSA dev'essere selezionata tra le opzioni disponibili all'interno del Q\_CHIUSO\_OPZIONI
- 17) Quando uno STUDENTE invia un messaggio PRIVATO ha come destinatario un PROFESSORE
- 18) Quando un PROFESSORE invia un BROADCAST viene ricevuto da tutti gli STUDENTE
- 19) Le tabelle di esercizio sono fisicamente tabelle SQL all'interno del database
- 20) In RISPOSTA la tripla (email\_studente, id\_quesito, timestamp) è UNIQUE
- 21) In QUESITO la coppia (test\_associato, numero\_quesito) è UNIQUE
- 22) In TAB\_ATT la coppia (nome\_tabella, nome\_attributo) è UNIQUE e viene usata come chiave dal dizionario VINCOLO

## 3. Progettazione logica

### 3.1 Ristrutturazione del diagramma concettuale

## QUESITO

Ho deciso di sostituire la generalizzazione di QUESITO con delle relazioni tra l'entità padre QUESITO e le figlie Q\_CHIUSO\_OPZIONE e Q\_APERTO\_SOLUZIONE, accorpendo le opzioni e le soluzioni nelle due tabelle figlie, non introducendo valori NULL, ma solo un campo per distinguere più rapidamente tra un tipo\_quesito e l'altro.

## **RISPOSTA**

Ho eseguito la stessa operazione per la generalizzazione di RISPOSTA, ovvero sostituendo la generalizzazione con due relazioni tra la tabella padre RISPOSTA e le due figlie R\_CHIUSA e R\_APERTA e introducendo un elemento per distinguere rapidamente il tipo\_risposta specifico del figlio, ma senza introdurre valori NULL.

R\_CHIUSA mantiene il suo collegamento con la tabella delle opzioni di un quesito chiuso.

## **UTENTE**

La stessa operazione è stata fatta per la tabella UTENTE, la sua generalizzazione è stata sostituita da relazioni padre e figli, con l'introduzione di un valore tipo\_utente per selezionare più rapidamente da una o dall'altra tabella figlia.

## **3.3 Ristrutturazione**

### **SVOLGIMENTO\_TEST**

Ho deciso di trasformare la relazione SVOLGIMENTO tra STUDENTE e TEST in una tabella SVOLGIMENTO\_TEST, poiché questa entità contiene informazioni che sono esterne alle occorrenze di uno STUDENTE o di un TEST, come l'inserimento della prima e ultima RISPOSTA e lo stato del TEST per quello specifico studente. Questo è stato fatto anche per non introdurre valori NULL data la relazione (0,N - 0,N) tra STUDENTE e TEST.

### **QUESITI-TABELLA**

È stata svolta la stessa operazione per la relazione ASSOCIAZIONE\_QT, ovvero il dizionario che mette in collegamento gli specifici quesiti di un test e le tabelle di esercizio a cui fanno riferimento, dato che la relazione nello schema concettuale era (0,N - 0,N) ho optato per trasformare la relazione in una tabella a sé, non introducendo valori NULL.

## **VINCOLO**

La relazione tra più attributi di diverse tabella di esercizi è stata trasformata in una tabella (VINCOLO), per non introdurre valori NULL, mantenendo la coerenza con le scelte precedenti.

## 3.4 Analisi delle ridondanze

### Costo operativo con ridondanza

- Aggiungere una nuova risposta ad un quesito esistente (10 volte/mese, interattiva)

$$OP1r = 10 * 1 * (2 * 3 + 0) = 10 * 6 = 60$$

- Rimuovere un quesito e tutte le risposte ottenute (2 volte/mese, batch)

$$OP2r = 2 * 1/2 * (2 * (1 + 2 * 10) + 0) = 2 * (1 + 20) = 42$$

- Visualizzare tutti gli utenti presenti nella piattaforma (1 volte/mese, batch)

$$OP3r = 1 * 1/2 * (2 * 0 + 50) = 25$$

- Contare il numero di risposte per ciascun quesito presente nella piattaforma (2 volte/mese, interattiva)

$$OP4r = 2 * 1 * (2 * 0 + 20) = 40$$

Costo totale con ridondanza:

$$SUMr = OP1r + OP2r + OP3r + OP4r = 60 + 42 + 25 + 40 = 167$$

### Costo operativo senza ridondanza

#### Speed up

- Aggiungere una nuova risposta ad un quesito esistente (10 volte/mese, interattiva)

$$OP1sr = 10 * 1 * (2 * 2 + 0) = 10 * 4 = 40$$

- Rimuovere un quesito e tutte le risposte ottenute (2 volte/mese, batch)

$$OP2sr = 2 * 1/2 * (2 * (1 + 2 * 10) + 0) = 2 * (1 + 20) = 42$$

- Visualizzare tutti gli utenti presenti nella piattaforma (1 volta/mese, batch)

$$OP3sr = 1 * 1/2 * (2 * 0 + 50) = 25$$

- Contare il numero di risposte per ciascun quesito presente nella piattaforma (2 volte/mese, interattiva)

$$OP4sr = 2 * 1 * (0 + 20 * 10) = 2 * 200 = 400$$

Costo totale senza ridondanza:

$$SUMsr = OP1sr + OP2sr + OP3sr + OP4sr = 40 + 42 + 25 + 400 = 507$$

Speed up:

$$SpeedUp = (SUMsr/SUMr) = 507/167 = 3.036 \Rightarrow 3$$

## Occupazione di memoria

La ridondanza è un INT, quindi il suo spazio di memoria è di 4 Byte per ogni record della tabella QUESITI, ovvero 20.

memoria (SUMsr) = x

memoria (SUMr) = x + 20 \* 4B = x + 80B

## Conclusione

Con la ridondanza si ha uno Speed Up pari a 3 a fronte di soli 80B di memoria aggiuntiva occupata, quindi è più efficiente mantenere la ridondanza.

## 3.5 Schema logico

### 3.5.1 Tabelle e vincoli di chiave primaria

BROADCAST (id\_messaggio, mittente)  
 FOTO\_DEL\_TEST (foto, test\_associato)  
 MESSAGGIO (id, titolo, testo, data\_inserimento, test\_associato)  
 MESSAGGIO\_PRIVATO(id\_messaggio, mittente, destinatario)  
 VINCOLO(nome\_tabella, nome\_attributo\_tabella\_vincolata, attributo\_vincolato)  
 PROFESSORE (email\_professore, dipartimento, corso)  
 QUESITI\_TABELLA (ID, id\_quesito, nome\_tabella)  
 QUESITO (ID, numero\_quesito, test\_associato, descrizione, livello\_difficolta, numero\_risposte, tipo\_quesito)  
 QUESITO\_APERTO\_SOLUZIONE (id\_quesito, id\_soluzione, soluzione\_professore)  
 QUESITO\_CHIUSO\_OPZIONE (id\_quesito, numero\_opzione, testo, is\_corretta)  
 RISPOSTA (ID, id\_quesito, email\_studente, TIMESTAMP, tipo\_risposta, esito)  
 RISPOSTA\_QUESITO\_APERTO (id\_risposta, risposta)  
 RISPOSTA\_QUESITO\_CHIUSO (id\_risposta, opzione\_scelta)  
 STUDENTE (email\_studente, matricola, anno\_immatricolazione)  
 SVOLGIMENTO\_TEST (titolo\_test, email\_studente, data\_inizio, data\_fine, stato)  
 TABELLA DELLE TABELLE (nome\_tabella, data\_creazione, num\_righe, creatore)  
 TAB\_ATT (ID, nome\_tabella, nome\_attributo, tipo\_attributo, key\_part)  
 TEST (titolo, dataCreazione, VisualizzaRisposte, email\_professore)  
 UTENTE (email, nome, cognome, PASSWORD, tipo\_utente, telefono)

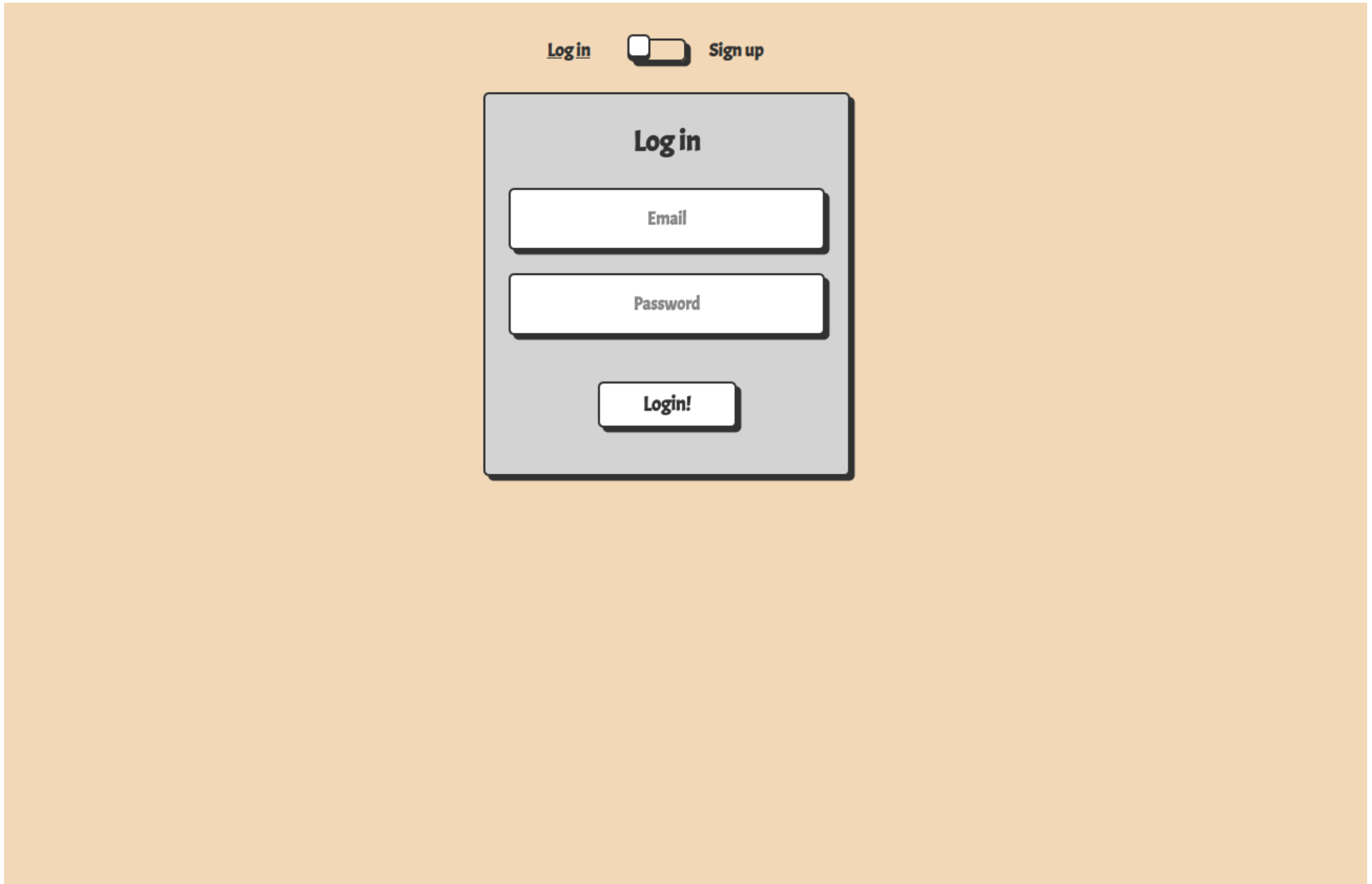
### 3.5.2 Tabelle e vincoli di chiave esterna

```
BROADCAST.id_messaggio => MESSAGGIO.id  
BROADCAST.mittente => PROFESSORE.email_professore  
VINCOLO.nome_tabella => TAB_ATT.nome_tabella  
VINCOLO.nome_attributo => TAB_ATT.nome_attributo  
VINCOLO.tabella_vincolata => TAB_ATT.nome_tabella  
VINCOLO.attributo_vincolato => TAB_ATT.nome_attributo  
FOTO_TEST.test_associato => TEST.titolo  
MESSAGGIO.test_associato => TEST.titolo  
MESSAGGIO_PRIVATO.id_messaggio => MESSAGGIO.id  
MESSAGGIO_PRIVATO.mittente => STUDENTE.email_studente  
MESSAGGIO_PRIVATO.destinatario => PROFESSORE.email_professore  
PROFESSORE.email_professore => UTENTE.email  
QUESITI_TABELLA.id_quesito => QUESITO.ID  
QUESITI_TABELLA.nome_tabella => TABELLA DELLE TABELLE.nome_tabella  
QUESITO.test_associato => TEST.titolo  
QUESITO_APERTO_SOLUZIONE.id_quesito => QUESITO.ID  
QUESITO_CHIUSO_OPZIONE.id_quesito => QUESITO.ID  
RISPOSTA.id_quesito => QUESITO.ID  
RISPOSTA.email_studente => STUDENTE.email_studente  
RISPOSTA_QUESITO_APERTO.id_risposta => RISPOSTA.ID  
RISPOSTA_QUESITO_CHIUSO.id_risposta => RISPOSTA.ID  
RISPOSTA_QUESITO_CHIUSO.opzione_scelta => QUESITO_CHIUSO_OPZIONE.numero_opzione  
STUDENTE.email_studente => UTENTE.email  
SVOLGIMENTO_TEST.titolo_test => TEST.titolo  
SVOLGIMENTO_TEST.email_studente => STUDENTE.email_studente  
TABELLA DELLE TABELLE.creatore => PROFESSORE.email_professore  
TAB_ATT.nome_tabella => TABELLA DELLE TABELLE.nome_tabella  
TEST.email_professore => PROFESSORE.email_professore
```

## 4. Descrizione e funzionamento della piattaforma

### 4.1 Login

La piattaforma mostra una classica interfaccia di login, con un campo "email" e un campo "password".



The image shows a login interface on a light orange background. At the top, there are two links: [Login](#) (underlined) and [Sign up](#) (with a small icon). Below these is a gray box containing the title **Log in**, an  field labeled **Email**, an  field labeled **Password**, and a **Login!** button.

## 4.2 Signup/registrazione



Cliccando sullo slider in sopra al widget di login, è possibile vedere l'interfaccia per registrarsi alla piattaforma.

[Log in](#)



[Sign up](#)

## Sign up

Studente ☐

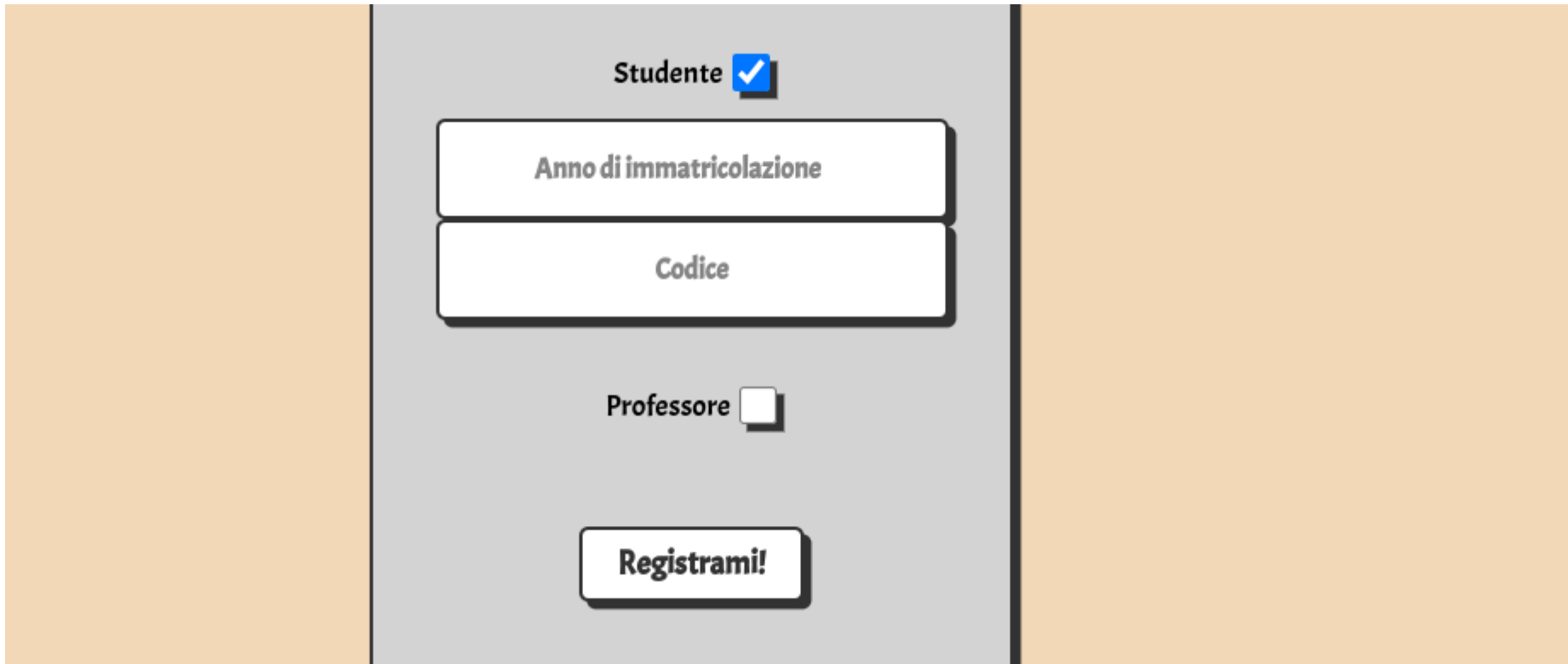
Professore ☐

**Registrami!**

Da questa schermata si dovrà poi selezionare un ruolo tra "Studente" e "Professore"

## 4.2.1 Registrazione studente

Lo studente deve inserire il suo anno di immatricolazione e il suo codice identificativo.



The image shows a registration form for a student. The form is centered on a light gray background, flanked by two vertical orange bars. At the top, the word "Studente" is followed by a blue checkmark icon. Below this, there are two white input fields with black borders. The first field is labeled "Anno di immatricolazione" and the second field is labeled "Codice". Below these fields, the word "Professore" is followed by a white square icon. At the bottom, there is a white button with a black border labeled "Registrami!".

Studente ☒

Anno di immatricolazione

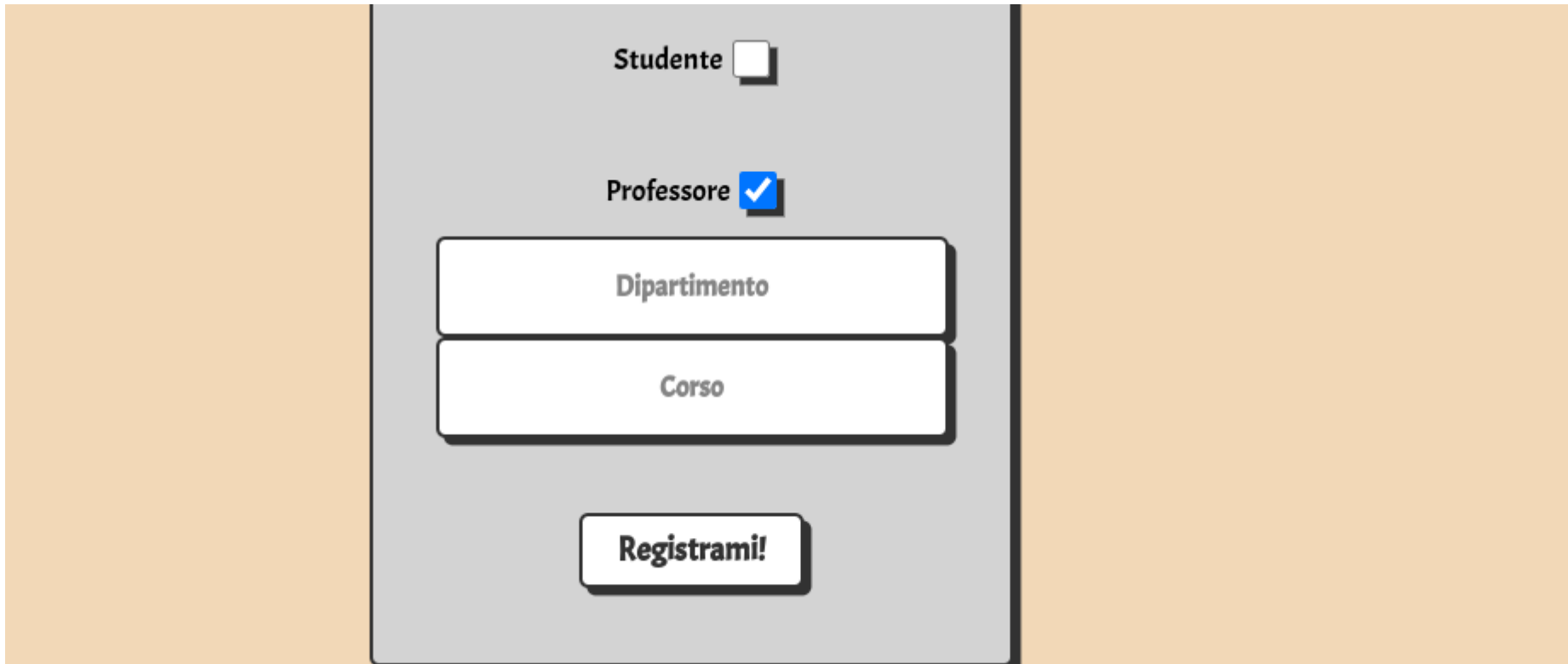
Codice

Professore ☐

**Registrami!**

## 4.2.2 Registrazione professore

Il professore, invece, deve inserire il suo dipartimento e il suo corso.



The image shows a registration form for a professor. It is a central gray rectangle with rounded corners, set against a light orange background. At the top, there are two options: 'Studente' with an unchecked checkbox and 'Professore' with a checked checkbox. Below these are two stacked white input fields with black borders, labeled 'Dipartimento' and 'Corso'. At the bottom of the form is a white button with a black border and the text 'Registrami!'.

## 4.3 Professore

Una volta fatto l'accesso alla piattaforma, il professore si trova davanti ad una schermata da cui può scegliere le sue prossime azioni, ovvero:

- Creare un test
- Creare una tabella
- Visualizzare le classifiche
- Interagire con le proprie informazioni:
  - Uscire dalla piattaforma cliccando l'icona in alto a sinistra
  - Visualizzare le informazioni personali cliccando l'icona in alto a destra



**Buongiorno prof. Oak**



<p><b>Crea un test</b></p> <p>Crea</p>	<p><b>Vai a creazione tabella</b></p> <p>Crea tabella</p>	<p><b>Vai a classifiche</b></p> <p>Classifiche</p>
--	---	--

#### 4.3.1 Creazione di una tabella

Se la scelta nella pagina del professore è stata quella di creare una tabella, allora ci si sposterà nell'apposita pagina.



## **Crea una tabella di esercizio**

In questo modulo si inserisce il nome della tabella che si vuole creare e il numero di attributi



**Crea una tabella di esercizio**

TIPI\_POKEMON

1

Crea

Tipo

Tipo Attributo 1:  
VARCHAR ▾

Primary Key ☒

Foreign Key ☐

Riempi la tabella

Una volta che la tabella è stata creata, si viene reindirizzati alla schermata da cui si possono aggiungere elementi.



**Inserisci valori**

Tabella: TIPI\_POKEMON

Tipo

Inserisci \*Tipo\*

Aggiungi riga

Dopo aver inserito il/i valori all'interno dell'apposito spazio, si procede cliccando "Aggiungi Riga" e la riga sarà, quindi, aggiunta.



**Inserisci valori**

Tabella: TIPI\_POKEMON

Tipo
Fuoco

Inserisci \*Tipo\*

Aggiungi riga

Attenzione, bisogna rispettare i vincoli di integrità che vengono mostrati in caso di esistenza di dipendenza verso altre tabelle create in precedenza.

Ad esempio, nella seguente immagine rappresento come, dopo aver creato la tabella POKEMON, ci sia un vincolo da POKEMON.tipo a TIPO\_POKEMON.Tipo





## Inserisci valori

[Mostra vincoli](#)**Attributo in POKEMON****Reference**

POKEMON.tipo ==&gt;

TIPI\_POKEMON.TIPO

**Tabella: POKEMON**

nome	tipo	PS
Chimchar	Fire	15
Dragonite	Dragon	30
Electabuzz	Electric	30
Machop	Fighting	20
Pikachu	Electric	10
Piplup	Water	8
Staraptor	Flying	14
Turtwig	Grass	8
Inserisci "nome"	Inserisci "tipo"	Inserisci "PS"

[Aggiungi riga](#)**Tabella: TIPI\_POKEMON**

TIPO
Dragon
Electric
Fighting
Fire
Flying
Grass
Ground
Ice
Normal
Poison
Water

## Inserimento dati in tabella

Sarà poi possibile per il professore inserire dei valori in una tabella a partire dalla schermata principale.



**Buongiorno prof. Oak**



**Crea un test**

**Crea**

**Vai a creazione tabella**

**Crea tabella**

**Vai a classifiche**

**Classifiche**

**Inserisci valori in tabella**

POKEMON ▾

**Vai**


### 4.3.2 Crea un test

Se è stato selezionato il tasto "Crea test", il professore potrà creare un nuovo test inserendo il nome all'interno dell'apposita sezione e interagendo con il tasto Crea.

È previsto anche l'inserimento di un'eventuale foto come copertina del test.



## Crea test



## Aggiungi i quesiti

Dopo aver creato il test si possono aggiungere i quesiti scegliendo anche eventuali tabelle dall'apposita sezione (se ne sono state create in anticipo).

È inoltre possibile eliminare il test, sia prima che dopo l'aggiunta di quesiti (in caso di esistenza di quesiti, verranno eliminati anche loro e tutte le risposte degli studenti ad essi associate)



## TEST POKEMON



### Aggiungi quesito

Descrizione:

Descrizione

Difficoltà:

Basso ▾

Aperto ☐

Chiuso ☐

Tabelle di riferimento:

POKEMON  
TIPI\_POKEMON

Aggiungi quesito

Per creare un quesito è necessario riempire gli appositi campi e selezionare la "Difficoltà" e se il quesito in questione è aperto o chiuso, per poter poi aggiungere le soluzioni (in caso di quesito aperto) o le opzioni (in caso di quesito chiuso).

È possibile associare più tabelle allo stesso quesito.



## TEST POKEMON



### Aggiungi quesito

Descrizione:

Pikachu è tipo acqua

Difficoltà:

Basso ▾

Aperto ☐

Chiuso ☒

Aggiungi opzione

Rimuovi opzione

Vero

Corretta

☐

Falso

Corretta



☒


Tabelle di riferimento:

POKEMON
TIPI_POKEMON

Aggiungi quesito

Una volta creato il quesito sarà possibile vederlo ed eliminarlo (eliminando a cascata tutte le risposte degli studenti per quel quesito)



TEST POKEMON

### Aggiungi quesito

Descrizione:

Descrizione

Difficoltà:

Basso ▾


Aperto ☐

Chiuso ☐

Tabelle di riferimento:

POKEMON  
TIPI\_POKEMON

Aggiungi quesito

Numero quesito	Descrizione	Difficoltà	Tipo	Tabelle di riferimento	Elimina quesito
1	Pikachu è tipo acqua	BASSO	CHIUSO	POKEMON, TIPI_POKEMON	

### 4.3.3 Modifica test

Una volta che sarà creato un test, sarà possibile accedere alla sua modifica dalla schermata principale, selezionando il test a cui si vogliono apportare modifiche.



Buongiorno prof. Oak



Crea un test

Crea

Modifica test

Seleziona un test ▾

Modifica

Concludi test

Seleziona un test ▾

Concludi test

Vai ai messaggi

Messaggi

Vai a creazione tabella

Crea tabella

Vai a classifiche

Classifiche

Inserisci valori in tabella

POKEMON ▾

Vai

## Concludi test

Dopo averlo creato, il professore può sempre decidere di concludere il test, in modo che gli studenti possano vedere le opzioni corrette e le soluzioni



Buongiorno prof. Oak



Crea un test

Crea

Modifica test

Seleziona un test ▾

Modifica

Concludi test

Seleziona un test ▾

Concludi test

Vai ai messaggi

Messaggi

Vai a creazione tabella

Crea tabella

Vai a classifiche

Classifiche

Inserisci valori in tabella

POKEMON ▾

Vai

#### 4.3.4 Invio di messaggi



Dopo aver creato un test, il professore può inviare messaggi a tutti gli studenti riferendosi ad uno specifico test.



**Invia un messaggio agli studenti**

**Invia un messaggio**

Saluti



Ciao a tutti gli studenti, io sono il professor. Oak

Test associato  
Test pokemon ▾

Invia

**Classifiche**

Il professore dalla propria homepage può sempre andare nella schermata delle classifiche per vedere le classifiche di tutti gli studenti.



## Classifiche

### Test completati

Matricola	Test completati
00970758	0
00123456	0
00987654	0
00567890	0

### Precisione delle risposte

Matricola	Percentuale risposte corrette
00567890	66.67%
00970758	0%
00123456	0%
00987654	0%

### Quesiti con maggior numero di risposte

Test	Quesito	Numero di risposte
Test pokemon	3	2
Test pokemon	1	1
Test pokemon	2	1

## 4.4 Studente

Lo studente dalla sua homepage, può vedere tutti i test che può svolgere e vede il loro stato attuale (ovvero se ha già inserito una risposta o è concluso)



#### 4.4.1 Svolgimento di un test

Per svolgere un test, lo studente deve scegliere uno dei quesiti che non è ancora stato concluso e verrà portato su una schermata di questo tipo.



## Esegui: TEST POKEMON

1. Pikachu è tipo acqua

- ☐ Vero  
☐ Falso

2. Seleziona il NOME dei pokemon di tipo 'Electric'

3. Seleziona il nome di tutti i tipo Water oppure Fire

Invia risposte

### Vincoli di integrità

POKEMON.tipo->TIPI\_POKEMON.TIPO

Tabella: POKEMON

nome	tipo	PS	is_legendary
Chimchar	Fire	15	0
Dragonite	Dragon	30	0
Electabuzz	Electric	30	0
Machop	Fighting	20	0
Pikachu	Electric	10	0
Piplup	Water	8	0
Staraptor	Flying	14	0
Turtwig	Grass	8	0

Tabella: TIPI\_POKEMON

TIPO
Dragon
Electric
Fighting
Fire
Flying
Grass
Ground
Ice
Normal
Poison
Water

### 4.4.2 Risultati

Una volta che il test viene considerato concluso (ovvero se lo studente ha inserito tutte le risposte corrette oppure il professore ha scelto di mostrare le risposte), lo studente potrà vedere i risultati per il suo test nell'apposita pagina, raggiungibile dalla sua homepage.



**Buongiorno Ash**



**Vai ai messaggi**

**Messaggi**

**Vai alle classifiche**

**Classifiche**

**Visualizza i tuoi test**

**Risultati**

**CITTÀ POKEMON**

**Il test è in completamento**

Data inizio: 2024-05-01 17:51:36

**Completa**

**POKEMON LEGGENDARI**

**Svolgi**

**TEST POKEMON**

**Il test è concluso**

In data: 2024-05-01 17:51:58

**Risultati**

Le risposte del professore, però verranno visualizzate solamente dopo che il professore avrà abilitato questa funzione.

**Risposte non visualizzabili**



## Test conclusi

Test pokemon				
Numero quesito	Data	Risposta dello studente	Risposta del professore	Esito
1	2024-05-01	2	-	GIUSTA
2	2024-05-01	SELECT nome FROM POKEMON WHERE tipo = 'Electric'	-	GIUSTA
3	2024-05-01	SELECT nome FROM POKEMON WHERE (tipo = 'Water'    tipo = 'Fire')	-	GIUSTA

Risposte visualizzabili



## Test conclusi

Test pokemon				
Numero quesito	Data	Risposta dello studente	Risposta del professore	Esito
1	2024-05-01	2	2	GIUSTA
2	2024-05-01	SELECT nome FROM POKEMON WHERE tipo = ' Electric '	SELECT nome FROM POKEMON WHERE tipo = ' Electric '	GIUSTA
3	2024-05-01	SELECT nome FROM POKEMON WHERE (tipo = ' Water '    tipo = ' Fire ' )	SELECT nome FROM POKEMON WHERE (tipo = ' Water '    tipo = ' Fire ' )	GIUSTA

### 4.4.3 Classifiche

Anche lo studente può vedere le classifiche e il suo numero di matricola verrà evidenziato per mostrargli più facilmente il proprio posizionamento



## Classifiche

### Test completati

Matricola	Test completati
00567890	2
00970758	1
00123456	1
00987654	1

### Precisione delle risposte

Matricola	Percentuale risposte corrette
00567890	100%
00970758	0%
00123456	0%
00987654	0%

### Quesiti con maggior numero di risposte

Test	Quesito	Numero di risposte
Test pokemon	3	2
Test pokemon	1	1
Test pokemon	2	1
Città pokemon	1	1

## 4.5 Logout

Selezionando il pulsante in alto a sinistra, si effettuerà il logout e si verrà riportati alla pagina di login.

## 5. Codice SQL

```
DROP DATABASE IF EXISTS POKEDB;
```



```
CREATE DATABASE IF NOT EXISTS POKEDB DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;
```

```
USE POKEDB;
```

```
-- Tabella Utente
```

```
CREATE TABLE IF NOT EXISTS
```

```
UTENTE (
```

```
email VARCHAR(100) PRIMARY KEY ,
```

```
nome VARCHAR(50) NOT NULL ,
```

```
cognome VARCHAR(50) NOT NULL ,
```

```
PASSWORD VARCHAR(100) NOT NULL ,
```

```
tipo_utente ENUM('STUDENTE', 'PROFESSORE') DEFAULT 'STUDENTE' NOT NULL,
```

```
telefono VARCHAR(20)
```

```
) ENGINE = InnoDB DEFAULT CHARSET = utf8 COLLATE = utf8_general_ci;
```

-- Tabella Studente

CREATE TABLE IF NOT EXISTS

STUDENTE (

email\_studente VARCHAR(100) PRIMARY KEY ,

matricola VARCHAR(16) NOT NULL ,

anno\_immatricolazione INT NOT NULL ,

FOREIGN KEY (email\_studente) REFERENCES UTENTE (email) ON DELETE CASCADE

);

-- Tabella Professore

CREATE TABLE IF NOT EXISTS

PROFESSORE (

email\_professore VARCHAR(100) PRIMARY KEY ,

dipartimento VARCHAR(100) NOT NULL ,

```
corso VARCHAR(100) NOT NULL ,

FOREIGN KEY (email_professore) REFERENCES UTENTE (email) ON DELETE CASCADE

);


-- crea tabella delle TABELLE create dai PROFESSORI

CREATE TABLE IF NOT EXISTS

TABELLA DELLE TABELLE (

nome_tabella VARCHAR(20) PRIMARY KEY ,

data_creazione DATETIME DEFAULT NOW() NOT NULL ,

num_righe INT DEFAULT 0 NOT NULL ,

creatore VARCHAR(100) NOT NULL ,

FOREIGN KEY (creatore) REFERENCES PROFESSORE (email_professore) ON DELETE CASCADE

);


-- Inserimento dati nella tabella UTENTE
```

```
INSERT INTO
```

```
UTENTE (
```

```
email ,
```

```
nome ,
```

```
cognome ,
```

```
PASSWORD ,
```

```
telefono ,
```

```
tipo_utente
```

```
)
```

```
VALUES
```

```
(
```

```
'studentel@unibo.it',
```

```
'Mario' ,
```

```
'Rossi' ,
```

```
'1234' ,
```

```
'12324567' ,  
  
'STUDENTE'  
  
) ,  
  
(  
  
'studente2@unibo.it',  
  
'Luca' ,  
  
'Bianchi' ,  
  
'1234' ,  
  
'12324567' ,  
  
'STUDENTE'  
  
) ,  
  
(  
  
'vincenzo.scollo@unibo.it',  
  
'Anna' ,  
  
'Verdi' ,
```

```
'1234' ,  
  
'12324567' ,  
  
'PROFESSORE'  
  
) ,  
  
(  
  
'mariagrazia.fabbri@unibo.it',  
  
'Carlo' ,  
  
'Neri' ,  
  
'1234' ,  
  
'12324567' ,  
  
'PROFESSORE'  
  
) ,  
  
(  
  
'simosamoggia@gmail.com',  
  
'Simone' ,
```

```
'Samoggia' ,  
  
'1234' ,  
  
'12324567' ,  
  
'STUDENTE'  
  
) ,  
  
(  
  
'professore@unibo.it',  
  
'Professor' ,  
  
'Oak' ,  
  
'1234' ,  
  
'12324567' ,  
  
'PROFESSORE'  
  
) ,  
  
(  
  
'studente@unibo.it',
```

```
'Ash' ,

'Ketchum' ,

'1234' ,

'12324567' ,

'STUDENTE'

);


-- Inserimento dati nella tabella Studente

INSERT INTO

STUDENTE (email_studente, anno_immatricolazione, matricola)

VALUES

('studente1@unibo.it', 2019, '00123456') ,

('studente2@unibo.it', 2020, '00987654') ,

('simosamoggia@gmail.com', 2020, '00970758'),

('studente@unibo.it', 2020, '00567890');
```



```
-- Inserimento dati nella tabella Professore

INSERT INTO

PROFESSORE (email_professore, dipartimento, corso)

VALUES

(

'vincenzo.scollo@unibo.it',

'Informatica' ,

'scienze applicate'

),

(

'mariagrazia.fabbri@unibo.it',

'Matematica' ,

'scienze applicate'

),
```

```
(  
  
    'professore@unibo.it',  
  
    'Matematica' ,  
  
    'Scienze applicate'  
  
);  
  
  
-- Procedura di registrazione studente  
  
DELIMITER $$  
  
CREATE PROCEDURE IF NOT EXISTS authenticateUser (  
  
    IN p_email VARCHAR(100) ,  
  
    IN p_password VARCHAR(100) ,  
  
    OUT p_authenticated TINYINT(1) ,  
  
    OUT p_tipo_utente ENUM('STUDENTE', 'PROFESSORE'),  
  
    OUT p_nome_utente VARCHAR(50) ,  
  
    OUT p_cognome_utente VARCHAR(50)
```

```
) BEGIN DECLARE user_count INT;
```

```
DECLARE tipo ENUM('STUDENTE', 'PROFESSORE');
```

```
DECLARE nome_utente VARCHAR(50);
```

```
DECLARE cognome_utente VARCHAR(50);
```

```
-- Controlla se le credenziali sono valide e recupera il tipo di utente, nome e cognome
```

```
SELECT
```

```
COUNT(*) ,
```

```
tipo_utente ,
```

```
nome ,
```

```
cognome INTO user_count,
```

```
tipo ,
```

```
nome_utente ,

cognome_utente

FROM

UTENTE

WHERE

email = p_email

AND PASSWORD = p_password

GROUP BY

tipo_utente,

nome ,

cognome;


-- Se user_count è maggiore di 0, significa che le credenziali sono valide

IF user_count > 0 THEN

SET
```

```
p_authenticated = 1;
```

```
SET
```

```
p_tipo_utente = tipo;
```

```
SET
```

```
p_nome_utente = nome_utente;
```

```
SET
```

```
p_cognome_utente = cognome_utente;
```

```
ELSE
```

```
SET
```

```
p_authenticated = 0;
```

SET

p\_tipo\_utente = NULL;

SET

p\_nome\_utente = NULL;

SET

p\_cognome\_utente = NULL;

END IF;

END \$\$ DELIMITER;

-- Procedura di registrazione studente

DELIMITER \$\$

```
CREATE PROCEDURE IF NOT EXISTS InserisciNuovoStudente (  
  
    IN p_email VARCHAR(100) ,  
  
    IN p_nome VARCHAR(50) ,  
  
    IN p_cognome VARCHAR(50) ,  
  
    IN p_password VARCHAR(100) ,  
  
    IN p_matricola VARCHAR(16) ,  
  
    IN p_anno_immatricolazione INT,  
  
    IN p_telefono VARCHAR(20)  
  
    ) BEGIN  
  
    START TRANSACTION;  
  
  
    -- Inserisce l'utente nella tabella Utente  
  
    INSERT INTO  
  
    UTENTE (email, nome, cognome, PASSWORD, telefono)  
  
    VALUES
```

```
(  
  
p_email ,  
  
p_nome ,  
  
p_cognome ,  
  
p_password,  
  
p_telefono  
  
);  
  
  
-- Inserisce lo studente nella tabella Studente  
  
INSERT INTO  
  
STUDENTE (email_studente, matricola, anno_immatricolazione)  
  
VALUES  
  
(p_email, p_matricola, p_anno_immatricolazione);  
  
  
COMMIT;
```



```
END $$ DELIMITER;
```

```
-- Procedura di registrazione professore
```

```
DELIMITER $$
```

```
CREATE PROCEDURE IF NOT EXISTS InserisciNuovoProfessore (
```

```
IN p_email VARCHAR(100) ,
```

```
IN p_nome VARCHAR(50) ,
```

```
IN p_cognome VARCHAR(50) ,
```

```
IN p_password VARCHAR(100) ,
```

```
IN p_dipartimento VARCHAR(100),
```

```
IN p_corso VARCHAR(100) ,
```

```
IN p_telefono VARCHAR(20)
```

```
) BEGIN
```

```
START TRANSACTION;
```

```
-- Inserisce l'utente nella tabella Utente
```

```
INSERT INTO
```

```
UTENTE (email, nome, cognome, PASSWORD, telefono)
```

```
VALUES
```

```
(
```

```
p_email ,
```

```
p_nome ,
```

```
p_cognome ,
```

```
p_password,
```

```
p_telefono
```

```
);
```

```
-- Inserisce il professore nella tabella Professore
```

```
INSERT INTO
```

```
PROFESSORE (email_professore, dipartimento, corso)
```

```
VALUES
```

```
(p_email, p_dipartimento, p_corso);
```

```
COMMIT;
```

```
END $$ DELIMITER;
```

```
-- crea tabella dei TEST
```

```
CREATE TABLE IF NOT EXISTS
```

```
TEST (
```

```
titolo VARCHAR(100) PRIMARY KEY ,
```

```
dataCreazione DATETIME DEFAULT NOW() NOT NULL ,
```

```
VisualizzaRisposte TINYINT(1) DEFAULT 0 NOT NULL ,
```

```
email_professore VARCHAR(100) NOT NULL ,
```

```
-- TINYINT viene utilizzato come BOOLEAN, dato che MySQL non supporta il tipo BOOLEAN

FOREIGN KEY (email_professore) REFERENCES PROFESSORE (email_professore) ON DELETE CASCADE

);


-- crea tabella delle foto dei test

CREATE TABLE IF NOT EXISTS

FOTO_TEST (

foto LONGBLOB NOT NULL ,

test_associato VARCHAR(100) NOT NULL ,

PRIMARY KEY (test_associato) ,

FOREIGN KEY (test_associato) REFERENCES TEST (titolo) ON DELETE CASCADE

);


-- crea procedura per inserire un nuovo TEST

DELIMITER $$
```

```
CREATE PROCEDURE IF NOT EXISTS InserisciNuovoTest (  
  
    IN p_titolo VARCHAR(100) ,  
  
    IN p_email_professore VARCHAR(100)  
  
    ) BEGIN  
  
    START TRANSACTION;  
  
  
    -- Inserisce il test nella tabella Test  
  
    INSERT INTO  
  
    TEST (titolo, email_professore)  
  
    VALUES  
  
    (p_titolo, p_email_professore);  
  
  
    COMMIT;  
  
  
    END $$ DELIMITER;
```

```
-- crea procedura per inserire una nuova foto di un test
```

```
DELIMITER $$
```

```
CREATE PROCEDURE IF NOT EXISTS InserisciNuovaFotoTest (
```

```
IN p_foto LONGBLOB ,
```

```
IN p_test_associato VARCHAR(100)
```

```
) BEGIN
```

```
START TRANSACTION;
```

```
-- Inserisce la foto del test nella tabella Foto_test
```

```
INSERT INTO
```

```
FOTO_TEST (foto, test_associato)
```

```
VALUES
```

```
(p_foto, p_test_associato);
```

```
COMMIT;
```

```
END $$ DELIMITER;
```

```
-- procedura per restituire l'immagine di un TEST
```

```
DELIMITER $$
```

```
CREATE PROCEDURE IF NOT EXISTS RecuperaFotoTest (IN p_test_associato VARCHAR(100)) BEGIN
```

```
SELECT
```

```
foto
```

```
FROM
```

```
FOTO_TEST
```

```
WHERE
```

```
test_associato = p_test_associato;
```

```
END $$ DELIMITER;
```

```
-- crea tabella dei QUESITI

CREATE TABLE IF NOT EXISTS

QUESITO (

ID INT AUTO_INCREMENT NOT NULL ,

numero_quesito INT NOT NULL ,

test_associato VARCHAR(100) NOT NULL ,

descrizione TEXT NOT NULL ,

livello_difficolta ENUM('BASSO', 'MEDIO', 'ALTO') DEFAULT 'BASSO' NOT NULL,

numero_risposte INT NOT NULL DEFAULT 0 ,

tipo_quesito ENUM('APERTO', 'CHIUSO') DEFAULT 'APERTO' NOT NULL ,

PRIMARY KEY (ID) ,

FOREIGN KEY (test_associato) REFERENCES TEST (titolo) ON DELETE CASCADE ,

CONSTRAINT quesito_test UNIQUE (test_associato, numero_quesito)

);
```



```
-- OPZIONE QUESITO CHIUSO
```

```
CREATE TABLE IF NOT EXISTS
```

```
QUESITO_CHIUSO_OPZIONE (
```

```
id_quesito INT NOT NULL ,
```

```
numero_opzione INT NOT NULL, -- opzione a, opzione b ... opzione z
```

```
testo TEXT NOT NULL ,
```

```
is_corretta ENUM('TRUE', 'FALSE') DEFAULT 'FALSE' NOT NULL ,
```

```
PRIMARY KEY (numero_opzione, id_quesito) ,
```

```
FOREIGN KEY (id_quesito) REFERENCES QUESITO (ID) ON DELETE CASCADE
```

```
);
```

```
-- crea tabella quesito aperto
```

```
CREATE TABLE IF NOT EXISTS
```

```
QUESITO_APERTO_SOLUZIONE (
```

```
id_quesito INT NOT NULL ,

id_soluzione INT AUTO_INCREMENT NOT NULL, -- soluzione 1, soluzione 2 ... soluzione n

soluzione_professore TEXT NOT NULL ,

PRIMARY KEY (id_soluzione) ,

FOREIGN KEY (id_quesito) REFERENCES QUESITO (ID) ON DELETE CASCADE

);
```

```
CREATE TABLE IF NOT EXISTS
```

```
RISPOSTA (
```

```
ID INT AUTO_INCREMENT NOT NULL ,
```

```
id_quesito INT NOT NULL ,
```

```
email_studente VARCHAR(100) NOT NULL ,
```

```
TIMESTAMP TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
```

```
tipo_risposta ENUM('APERTA', 'CHIUSA') DEFAULT 'APERTA' NOT NULL ,
```

```
esito ENUM('GIUSTA', 'SBAGLIATA') DEFAULT 'SBAGLIATA' NOT NULL ,
```

```
PRIMARY KEY (ID) ,

FOREIGN KEY (id_quesito) REFERENCES QUESITO (ID) ON DELETE CASCADE ,

FOREIGN KEY (email_studente) REFERENCES STUDENTE (email_studente) ON DELETE CASCADE,

CONSTRAINT risposta_quesito UNIQUE (id_quesito, TIMESTAMP, email_studente)

);


-- crea tabelle delle risposte chiuse

CREATE TABLE IF NOT EXISTS

RISPOSTA_QUESITO_CHIUSO (

id_risposta INT NOT NULL ,

opzione_scelta INT NOT NULL ,

PRIMARY KEY (id_risposta) ,

FOREIGN KEY (id_risposta) REFERENCES RISPOSTA (ID) ON DELETE CASCADE ,

FOREIGN KEY (opzione_scelta) REFERENCES QUESITO_CHIUSO_OPZIONE (numero_opzione) ON DELETE CASCADE

);
```

```
-- crea tabella delle risposte aperte
```

```
CREATE TABLE IF NOT EXISTS
```

```
RISPOSTA_QUESITO_APERTO (
```

```
id_risposta INT NOT NULL ,
```

```
risposta TEXT NOT NULL ,
```

```
PRIMARY KEY (id_risposta) ,
```

```
FOREIGN KEY (id_risposta) REFERENCES RISPOSTA (ID) ON DELETE CASCADE
```

```
);
```

```
-- crea la stored procedure per inserire una nuova opzione per un quesito chiuso
```

```
DELIMITER $$
```

```
CREATE PROCEDURE IF NOT EXISTS InserisciNuovaOpzioneQuesitoChiuso (
```

```
IN p_numero_opzione INT ,
```

```
IN p_id_quesito INT ,
```

```
IN p_testo TEXT ,

IN p_is_corretta ENUM('TRUE', 'FALSE')

) BEGIN

-- Inserisce l'opzione nella tabella QUESITO_CHIUSO_OPZIONE

INSERT INTO

QUESITO_CHIUSO_OPZIONE (id_quesito, numero_opzione, testo, is_corretta)

VALUES

(

p_id_quesito ,

p_numero_opzione,

p_testo ,

p_is_corretta

);

COMMIT;
```

```
END $$ DELIMITER;
```

```
-- aggiorna il numero_risposte di un quesito quando viene aggiunto una nuova opzione
```

```
DELIMITER $$
```

```
CREATE TRIGGER IF NOT EXISTS update_numero_risposte_al_quesito AFTER
```

```
INSERT
```

```
ON RISPOSTA FOR EACH ROW BEGIN
```

```
UPDATE QUESITO
```

```
SET
```

```
numero_risposte = numero_risposte + 1
```

```
WHERE
```

```
QUESITO.ID = NEW.id_quesito;
```

```
END $$ DELIMITER;
```

```
-- crea la stored procedure per inserire una nuova soluzione per un quesito aperto
```

```
DELIMITER $$
```

```
CREATE PROCEDURE IF NOT EXISTS InserisciNuovaSoluzioneQuesitoAperto (
```

```
IN p_id_quesito INT ,
```

```
IN p_soluzione_professore TEXT
```

```
) BEGIN
```

```
START TRANSACTION;
```

```
-- Inserisce la soluzione nella tabella Quesito_aperto
```

```
INSERT INTO
```

```
QUESITO_APERTO_SOLUZIONE (id_quesito, soluzione_professore)
```

```
VALUES
```

```
(p_id_quesito, p_soluzione_professore);
```

```
COMMIT;
```

```
END $$ DELIMITER;
```

```
-- procedura per inserire un nuovo QUESITO
```

```
DELIMITER $$
```

```
CREATE PROCEDURE IF NOT EXISTS InserisciNuovoQuesito (
```

```
IN p_numero_quesito INT ,
```

```
IN p_test_associato VARCHAR(100) ,
```

```
IN p_descrizione TEXT ,
```

```
IN p_livello_difficolta ENUM('BASSO', 'MEDIO', 'ALTO'),
```

```
IN p_tipo_quesito ENUM('APERTO', 'CHIUSO') ,
```

```
OUT id_nuovo_quesito INT
```

```
) BEGIN
```

```
START TRANSACTION;
```



```
-- Inserisce il quesito nella tabella Quesito
```

```
INSERT INTO
```

```
QUESITO (
```

```
numero_quesito ,
```

```
test_associato ,
```

```
descrizione ,
```

```
livello_difficolta,
```

```
tipo_quesito
```

```
)
```

```
VALUES
```

```
(
```

```
p_numero_quesito ,
```

```
p_test_associato ,
```

```
p_descrizione ,
```

```
p_livello_difficolta,
```

```
p_tipo_quesito
```

```
);
```

```
-- Ottiene l'ID del quesito appena inserito
```

```
SELECT
```

```
LAST_INSERT_ID() INTO id_nuovo_quesito;
```

```
COMMIT;
```

```
END $$ DELIMITER;
```

```
-- -- crea tabella degli ATTRIBUTI delle TABELLE create dai PROFESSORI
```

```
CREATE TABLE IF NOT EXISTS
```

```
TAB_ATT (
```

```
ID INT AUTO_INCREMENT ,

nome_tabella VARCHAR(20) NOT NULL ,

nome_attributo VARCHAR(100) NOT NULL ,

tipo_attributo VARCHAR(15) NOT NULL ,

key_part ENUM('TRUE', 'FALSE') DEFAULT 'FALSE' NOT NULL ,

PRIMARY KEY (ID) ,

CONSTRAINT UNIQUE_TAB_ATT UNIQUE (nome_tabella, nome_attributo) ,

FOREIGN KEY (nome_tabella) REFERENCES TABELLA DELLE TABELLE (nome_tabella) ON DELETE CASCADE

);
```

```
-- -- crea tabella delle CHIAVI ESTERNE delle TABELLE create dai PROFESSORI
```

```
CREATE TABLE IF NOT EXISTS
```

```
VINCOLI (
```

```
nome_tabella VARCHAR(20) NOT NULL ,
```

```
nome_attributo VARCHAR(100) NOT NULL ,
```

```

tabella_vincolata VARCHAR(20) NOT NULL ,

attributo_vincolato VARCHAR(100) NOT NULL,

PRIMARY KEY (

nome_tabella ,

nome_attributo ,

tabella_vincolata ,

attributo_vincolato

) ,

FOREIGN KEY (nome_tabella, nome_attributo) REFERENCES TAB_ATT (nome_tabella, nome_attributo) ON DELETE CASCADE ,

FOREIGN KEY (tabella_vincolata, attributo_vincolato) REFERENCES TAB_ATT (nome_tabella, nome_attributo) ON DELETE CASCADE

);

```

create table

```

`TIPI_POKEMON` (

`TIPO` varchar(255) not null,

```

```
primary key (`TIPO`)
```

```
);
```

```
create table
```

```
`POKEMON` (
```

```
`PS` int not null ,
```

```
`nome` varchar(255) not null ,
```

```
`tipo` varchar(255) not null ,
```

```
`is_legendary` INT default 0 not null ,
```

```
primary key (`nome`) ,
```

```
foreign key (`tipo`) references `TIPI_POKEMON` (`TIPO`) ON DELETE CASCADE
```

```
);
```

```
INSERT INTO
```

```
`TIPI_POKEMON` (`TIPO`)
```

VALUES

```
('Dragon') ,  
  
('Electric'),  
  
('Fighting'),  
  
('Fire') ,  
  
('Flying') ,  
  
('Grass') ,  
  
('Ground') ,  
  
('Ice') ,  
  
('Normal') ,  
  
('Poison') ,  
  
('Water');
```

INSERT INTO

```
`POKEMON` (`PS`, `nome`, `tipo`, `is_legendary`)
```

VALUES

```
(15, 'Chimchar', 'Fire', 0) ,  
  
(30, 'Dragonite', 'Dragon', 0) ,  
  
(30, 'Electabuzz', 'Electric', 0),  
  
(20, 'Machop', 'Fighting', 0) ,  
  
(10, 'Pikachu', 'Electric', 0) ,  
  
(8, 'Piplup', 'Water', 0) ,  
  
(14, 'Staraptor', 'Flying', 0) ,  
  
(8, 'Turtwig', 'Grass', 0);
```

-- CREA tabella SVOLGIMENTO TEST

CREATE TABLE IF NOT EXISTS

SVOLGIMENTO\_TEST (

titolo\_test VARCHAR(100) NOT NULL ,

email\_studente VARCHAR(100) NOT NULL ,

```
data_inizio TIMESTAMP ,

data_fine TIMESTAMP ,

stato ENUM('APERTO', 'IN_COMPLETAMENTO', 'CONCLUSO') DEFAULT 'APERTO' NOT NULL ,

PRIMARY KEY (titolo_test, email_studente) ,

FOREIGN KEY (titolo_test) REFERENCES TEST (titolo) ON DELETE CASCADE ,

FOREIGN KEY (email_studente) REFERENCES STUDENTE (email_studente) ON DELETE CASCADE

);
```

```
-- tabella dei messaggi
```

```
CREATE TABLE IF NOT EXISTS
```

```
MESSAGGIO (
```

```
id INT AUTO_INCREMENT ,
```

```
titolo VARCHAR(100) NOT NULL ,
```

```
testo TEXT NOT NULL ,
```

```
data_inserimento DATETIME DEFAULT CURRENT_TIMESTAMP NOT NULL ,
```



```
test_associato VARCHAR(100) NOT NULL ,

PRIMARY KEY (id) ,

FOREIGN KEY (test_associato) REFERENCES TEST (titolo) ON DELETE CASCADE

);
```

```
-- messaggio che invia uno studente al professore
```

```
CREATE TABLE IF NOT EXISTS
```

```
MESSAGGIO_PRIVATO (
```

```
id_messaggio INT NOT NULL ,
```

```
mittente VARCHAR(100) NOT NULL ,
```

```
destinatario VARCHAR(100) NOT NULL ,
```

```
PRIMARY KEY (id_messaggio) ,
```

```
FOREIGN KEY (id_messaggio) REFERENCES MESSAGGIO (id) ON DELETE CASCADE ,
```

```
FOREIGN KEY (mittente) REFERENCES STUDENTE (email_studente) ON DELETE CASCADE ,
```

```
FOREIGN KEY (destinatario) REFERENCES PROFESSORE (email_professore) ON DELETE CASCADE
```

```
);
```

```
-- messaggio che invia un professore a tutti gli studenti
```

```
CREATE TABLE IF NOT EXISTS
```

```
BROADCAST (
```

```
id_messaggio INT NOT NULL ,
```

```
mittente VARCHAR(100) NOT NULL ,
```

```
PRIMARY KEY (id_messaggio) ,
```

```
FOREIGN KEY (id_messaggio) REFERENCES MESSAGGIO (id) ON DELETE CASCADE ,
```

```
FOREIGN KEY (mittente) REFERENCES PROFESSORE (email_professore) ON DELETE CASCADE
```

```
);
```

```
-- crea tabella dei quesiti-tabella
```

```
CREATE TABLE IF NOT EXISTS
```

```
QUESITI_TABELLA (
```

```
ID INT AUTO_INCREMENT ,

id_quesito INT NOT NULL ,

nome_tabella VARCHAR(20) NOT NULL ,

PRIMARY KEY (ID) ,

FOREIGN KEY (id_quesito) REFERENCES QUESITO (ID) ON DELETE CASCADE ,

FOREIGN KEY (nome_tabella) REFERENCES TABELLA DELLE TABELLE (nome_tabella) ON DELETE CASCADE,

CONSTRAINT UNIQUE_quesito_tabella UNIQUE (id_quesito, nome_tabella)

);
```

```
-- procedura per prendere il numero del nuovo quesito
```

```
DELIMITER $$
```

```
CREATE PROCEDURE GetNumeroNuovoQuesito (IN p_test_associato VARCHAR(100)) BEGIN
```

```
SELECT
```

```
numero_quesito
```

```
FROM
```

QUESITO

WHERE

test\_associato = p\_test\_associato

ORDER BY

numero\_quesito DESC

LIMIT

1;

END \$\$ DELIMITER;

-- riparti da qui

-- crea procedure che verifica se lo studente ha già concluso quel test

DELIMITER \$\$

CREATE PROCEDURE IF NOT EXISTS VerificaTestConcluso (

IN p\_email\_studente VARCHAR(100),

```
IN p_titolo_test VARCHAR(100) ,

OUT is_closed INT

) BEGIN

SELECT

COUNT(*) INTO is_closed

FROM

SVOLGIMENTO_TEST

WHERE

email_studente = p_email_studente

AND titolo_test = p_titolo_test

AND stato = 'CONCLUSO';

END $$ DELIMITER;

-- inserisci risposta a quesito chiuso
```

```
DELIMITER $$

CREATE PROCEDURE IF NOT EXISTS InserisciRispostaQuesitoChiuso (

IN p_id_quesito INT ,

IN p_email_studente VARCHAR(100) ,

IN p_opzione_scelta INT ,

IN p_esito ENUM('GIUSTA', 'SBAGLIATA')

) BEGIN DECLARE id_risposta INT;

DECLARE test_closed INT;

DECLARE p_test_associato VARCHAR(100);

START TRANSACTION;

-- Ottiene il test associato al quesito
```

```
SELECT
```

```
`QUESITO`.test_associato INTO p_test_associato
```

```
FROM
```

```
QUESITO
```

```
WHERE
```

```
ID = p_id_quesito;
```

```
-- Controlla se il test è già concluso
```

```
CALL VerificaTestConcluso (p_email_studente, p_test_associato, test_closed);
```

```
IF test_closed < 1 THEN
```

```
-- inserisce la risposta nella tabella Risposta
```

```
INSERT INTO
```

```
RISPOSTA (id_quesito, email_studente, tipo_risposta, esito)
```

```
VALUES
```

```
(p_id_quesito, p_email_studente, 'CHIUSA', p_esito);

-- Ottiene l'id della risposta appena inserita

SELECT

LAST_INSERT_ID() INTO id_risposta;

-- Inserisce la risposta nella tabella Risposta_quesito_chiuso

INSERT INTO

RISPOSTA_QUESITO_CHIUSO (id_risposta, opzione_scelta)

VALUES

(id_risposta, p_opzione_scelta);

END IF;

-- Esegue il commit della transazione
```



```
COMMIT;
```

```
END $$ DELIMITER;
```

```
DELIMITER $$
```

```
-- inserisci risposta a quesito aperto
```

```
CREATE PROCEDURE IF NOT EXISTS InserisciRispostaQuesitoAperto (
```

```
IN p_id_quesito INT ,
```

```
IN p_email_studente VARCHAR(100) ,
```

```
IN p_risposta TEXT ,
```

```
IN p_esito ENUM('GIUSTA', 'SBAGLIATA')
```

```
) BEGIN DECLARE id_risposta INT;
```

```
DECLARE test_closed INT;
```

```
DECLARE p_test_associato VARCHAR(100);
```

```
START TRANSACTION;
```

```
-- Ottiene il test associato al quesito
```

```
SELECT
```

```
`QUESITO`.test_associato INTO p_test_associato
```

```
FROM
```

```
QUESITO
```

```
WHERE
```

```
ID = p_id_quesito;
```

```
CALL VerificaTestConcluso (p_email_studente, p_test_associato, test_closed);
```

```
IF test_closed < 1 THEN
```

```
-- inserisce la risposta nella tabella Risposta

INSERT INTO

RISPOSTA (id_quesito, email_studente, tipo_risposta, esito)

VALUES

(p_id_quesito, p_email_studente, 'APERTA', p_esito);


-- Ottiene l'id della risposta appena inserita

SELECT

LAST_INSERT_ID() INTO id_risposta;


-- Inserisce la risposta nella tabella Risposta_quesito_aperto

INSERT INTO

RISPOSTA_QUESITO_APERTO (id_risposta, risposta)

VALUES

(id_risposta, p_risposta);
```

```
END IF;
```

```
COMMIT;
```

```
END $$ DELIMITER;
```

```
DELIMITER $$
```

```
CREATE PROCEDURE GetRisposteQuesiti (
```

```
IN p_test_associato VARCHAR(100),
```

```
IN p_email_studente VARCHAR(100)
```

```
) BEGIN
```

```
SELECT
```

```
r.id_quesito ,
```

```
q.numero_quesito ,
```

```
DATE(r.TIMESTAMP) AS in_data,

r.esito ,

r.tipo_risposta ,

q.ID AS `ID_Q`

FROM

RISPOSTA AS r

JOIN QUESITO q ON r.id_quesito = q.ID

WHERE

r.ID IN (

SELECT

MAX(r1.ID)

FROM

RISPOSTA r1

JOIN QUESITO q1 ON r1.id_quesito = q1.ID

WHERE
```

```
q1.test_associato = p_test_associato

AND r1.email_studente = p_email_studente

GROUP BY

q1.numero_quesito

)

GROUP BY

ID_Q ,

q.numero_quesito,

r.TIMESTAMP ,

r.esito ,

r.tipo_risposta

ORDER BY

q.numero_quesito ASC;

END $$ DELIMITER;
```

```
-- getAllTests
```

```
DELIMITER $$
```

```
CREATE PROCEDURE GetAllTests () BEGIN
```

```
SELECT
```

```
*
```

```
FROM
```

```
TEST;
```

```
END $$ DELIMITER;
```

```
-- get quesiti del test
```

```
DELIMITER $$
```

```
CREATE PROCEDURE GetQuesitiTest (IN p_test_associato VARCHAR(100)) BEGIN
```

```
SELECT
```

\*

FROM

QUESITO

WHERE

test\_associato = p\_test\_associato;

END \$\$ DELIMITER;

-- get opzioni quesito chiuso del test

DELIMITER \$\$

CREATE PROCEDURE GetOpzioniQuesitoChiuso (IN p\_id\_quesito INT) BEGIN

SELECT

\*

FROM

QUESITO\_CHIUSO\_OPZIONE



WHERE

QUESITO\_CHIUSO\_OPZIONE.id\_quesito = p\_id\_quesito;

END \$\$ DELIMITER;

-- prendi opzioni quesito chiuso vere

DELIMITER \$\$

CREATE PROCEDURE GetOpzioniCorrette (IN p\_id\_quesito INT) BEGIN

SELECT

\*

FROM

QUESITO\_CHIUSO\_OPZIONE

WHERE

id\_quesito = p\_id\_quesito

AND is\_corretta = 'TRUE';

```
END $$ DELIMITER;
```

```
-- prendi test del docente
```

```
DELIMITER $$
```

```
CREATE PROCEDURE GetTestsDelProfessore (IN p_email_professore VARCHAR(100)) BEGIN
```

```
SELECT
```

```
*
```

```
FROM
```

```
TEST
```

```
WHERE
```

```
email_professore = p_email_professore;
```

```
END $$ DELIMITER;
```

```
-- get risposta aperta from risposta

DELIMITER $$

DROP PROCEDURE IF EXISTS GetRispostaQuesitoAperto $$

CREATE PROCEDURE GetRispostaQuesitoAperto (

IN p_id_quesito INT ,

IN p_email_studente VARCHAR(100)

) BEGIN

SELECT

id_risposta,

risposta ,

esito ,

TIMESTAMP

FROM

RISPOSTA_QUESITO_APERTO as ra

JOIN RISPOSTA as r on ra.id_risposta = r.ID
```

WHERE

id\_risposta = (

SELECT

ID

FROM

RISPOSTA

WHERE

id\_quesito = p\_id\_quesito

AND email\_studente = p\_email\_studente

GROUP BY

ID

ORDER BY

ID DESC

LIMIT

1

```
);
```

```
END $$ DELIMITER;
```

```
-- get scelta quesito chiuso from risposta
```

```
DELIMITER $$
```

```
CREATE PROCEDURE GetSceltaQuesitoChiuso (
```

```
IN p_id_quesito INT ,
```

```
IN p_email_studente VARCHAR(100)
```

```
) BEGIN
```

```
SELECT
```

```
*
```

```
FROM
```

```
RISPOSTA_QUESITO_CHIUSO ra
```

```
JOIN RISPOSTA r ON r.ID = ra.id_risposta
```

WHERE

id\_risposta = (

SELECT

ID

FROM

RISPOSTA

WHERE

id\_quesito = p\_id\_quesito

AND email\_studente = p\_email\_studente

GROUP BY

ID

ORDER BY

ID DESC

LIMIT

1

```
);
```

```
END $$ DELIMITER;
```

```
-- inserisci il nuovo test creato all'interno di svolgimento_test per ogni studente con lo stato APERTO
```

```
DELIMITER $$
```

```
CREATE TRIGGER IF NOT EXISTS after_test_creation AFTER
```

```
INSERT
```

```
ON TEST FOR EACH ROW BEGIN
```

```
-- Inserire i record per tutti gli studenti nella tabella SVOLGIMENTO_TEST
```

```
INSERT INTO
```

```
SVOLGIMENTO_TEST (titolo_test, email_studente)
```

```
SELECT
```

```
NEW.titolo ,
```

```
email_studente
```

```
FROM
```

```
STUDENTE;
```

```
END $$ DELIMITER;
```

```
-- SONO QUI
```

```
-- trigger che si aziona quando un utente inserisce una risposta
```

```
-- la tabella svolgimento_test viene aggiornata e viene settata la data di inizio e lo stato IN_COMPLETAMENTO
```

```
DELIMITER $$
```

```
CREATE TRIGGER IF NOT EXISTS update_svolgimento_test AFTER
```

```
INSERT
```

```
ON RISPOSTA FOR EACH ROW BEGIN DECLARE is_prima_risposta INT;
```

```
-- prendi il test associato alla risposta
```

```
DECLARE p_test_associato VARCHAR(100);
```



```
SELECT
```

```
test_associato INTO p_test_associato
```

```
FROM
```

```
QUESITO
```

```
WHERE
```

```
ID = NEW.id_quesito;
```

```
SELECT
```

```
COUNT(*) INTO is_prima_risposta
```

```
FROM
```

```
RISPOSTA
```

```
JOIN QUESITO ON RISPOSTA.id_quesito = QUESITO.ID
```

```
WHERE
```

```
QUESITO.test_associato = p_test_associato
```

```
AND email_studente = NEW.email_studente;
```

```
IF is_prima_risposta = 1 THEN
```

```
UPDATE SVOLGIMENTO_TEST
```

```
SET
```

```
data_inizio = NOW() ,
```

```
stato = 'IN_COMPLETAMENTO'
```

```
WHERE
```

```
titolo_test = p_test_associato
```

```
AND email_studente = NEW.email_studente;
```

```
END IF;
```

```
-- Se tutte le ultime risposte inserite sono giuste, setta lo stato a CONCLUSO e la data di fine del test
```

```
IF(
```

```
SELECT
```

```
count(*)
```

```
FROM
```

```
RISPOSTA r
```

```
WHERE
```

```
r.esito = 'GIUSTA'
```

```
AND r.ID in (
```

```
SELECT
```

```
MAX(r1.ID)
```

```
FROM
```

```
RISPOSTA r1
```

```
JOIN QUESITO q ON r1.id_quesito = q.ID
```

```
WHERE
```

```
q.test_associato = p_test_associato
```

```
AND r1.email_studente = NEW.email_studente
```

```
GROUP BY
```

```
  r1.id_quesito
```

```
)
```

```
) = (
```

```
SELECT
```

```
  COUNT(*)
```

```
FROM
```

```
  QUESITO
```

```
WHERE
```

```
  test_associato = p_test_associato
```

```
) THEN
```

```
UPDATE SVOLGIMENTO_TEST
```

```
SET
```

```
  stato = 'CONCLUSO' ,
```

```
  data_fine = NEW.TIMESTAMP
```

WHERE

titolo\_test = p\_test\_associato

AND email\_studente = NEW.email\_studente;

END IF;

END \$\$ DELIMITER;

-- show results

DELIMITER \$\$

CREATE PROCEDURE MostraRisultati (IN p\_test\_associato VARCHAR(100)) BEGIN

UPDATE TEST

SET

VisualizzaRisposte = 1

WHERE

```
titolo = p_test_associato;
```

```
END $$ DELIMITER;
```

```
-- se il prof modifica il test impostando visualizzatest come true, allora contrassegna tutti i test come conclusi
```

```
DELIMITER $$
```

```
CREATE TRIGGER IF NOT EXISTS update_test_conclusi AFTER
```

```
UPDATE ON TEST FOR EACH ROW BEGIN
```

```
-- Se il professore ha impostato VisualizzaRisposte a 1, contrassegna tutti i test come CONCLUSI
```

```
IF NEW.VisualizzaRisposte = 1 THEN
```

```
UPDATE SVOLGIMENTO_TEST
```

```
SET
```

```
stato = 'CONCLUSO',
```

```
data_fine = NOW()
```

```
WHERE
```

```
titolo_test = NEW.titolo;
```

```
END IF;
```

```
END $$ DELIMITER;
```

```
-- studente invia messaggio
```

```
DELIMITER $$
```

```
CREATE PROCEDURE IF NOT EXISTS InviaMessaggioDaStudente (
```

```
IN p_titolo VARCHAR(100) ,
```

```
IN p_testo TEXT ,
```

```
IN p_test_associato VARCHAR(100),
```

```
IN p_mittente VARCHAR(100) ,
```

```
IN p_destinatario VARCHAR(100)
```

```
) BEGIN
```

```
START TRANSACTION;
```

```
-- Inserisce il messaggio nella tabella MESSAGGIO
```

```
INSERT INTO
```

```
MESSAGGIO (titolo, testo, test_associato)
```

```
VALUES
```

```
(p_titolo, p_testo, p_test_associato);
```

```
-- Inserisce il messaggio nella tabella MESSAGGIO_PRIVATO
```

```
INSERT INTO
```

```
MESSAGGIO_PRIVATO (id_messaggio, mittente, destinatario)
```

```
VALUES
```

```
(LAST_INSERT_ID(), p_mittente, p_destinatario);
```



```
COMMIT;
```

```
END $$ DELIMITER;
```

```
-- docente invia messaggio a tutti gli studenti
```

```
DELIMITER $$
```

```
CREATE PROCEDURE InviaMessaggioDaDocente (
```

```
IN p_mittente VARCHAR(100) ,
```

```
IN p_titolo VARCHAR(100) ,
```

```
IN p_testo TEXT ,
```

```
IN p_test_associato VARCHAR(100)
```

```
) BEGIN DECLARE last_id INT;
```

```
START TRANSACTION;
```

```
INSERT INTO
```

```
MESSAGGIO (titolo, testo, test_associato)
```

```
VALUES
```

```
(p_titolo, p_testo, p_test_associato);
```

```
SELECT
```

```
LAST_INSERT_ID() INTO last_id;
```

```
INSERT INTO
```

```
BROADCAST (id_messaggio, mittente)
```

```
VALUES
```

```
(last_id, p_mittente);
```

```
COMMIT;
```

```
END $$ DELIMITER;

-- get messaggi studente

DELIMITER $$

CREATE PROCEDURE GetMessaggiStudente (IN p_email_studente VARCHAR(100)) BEGIN

SELECT

m.id ,

m.titolo ,

m.testo ,

m.data_inserimento,

m.test_associato ,

b.mittente

FROM

MESSAGGIO as m

JOIN BROADCAST as b on b.id_messaggio = m.id;
```

```
END $$ DELIMITER;
```

```
-- get messaggi professore
```

```
DELIMITER $$
```

```
CREATE PROCEDURE GetMessaggiProfessore (IN p_email_professore VARCHAR(100)) BEGIN
```

```
SELECT
```

```
m.id ,
```

```
m.titolo ,
```

```
m.testo ,
```

```
m.data_inserimento,
```

```
m.test_associato ,
```

```
DM.mittente
```

```
FROM
```

```
MESSAGGIO as m ,
```

```
MESSAGGIO_PRIVATO as DM
```

```
WHERE
```

```
m.id = DM.id_messaggio
```

```
AND DM.destinatario = p_email_professore;
```

```
END $$ DELIMITER;
```

```
-- classifica test conclusi dagli studenti
```

```
CREATE VIEW
```

```
Classifica_test_completati AS
```

```
SELECT
```

```
s.matricola,
```

```
(
```

```
SELECT
```

```
COUNT(*)
```

```
FROM
```

```
SVOLGIMENTO_TEST st
```

```
WHERE
```

```
st.email_studente = s.email_studente
```

```
AND st.stato = 'CONCLUSO'
```

```
) AS Test_conclusi
```

```
FROM
```

```
STUDENTE s
```

```
GROUP BY
```

```
s.email_studente,
```

```
Test_conclusi
```

```
ORDER BY
```

```
Test_conclusi DESC;
```

```
# Visualizzare la classifica degli studenti, sulla base del numero di risposte corrette inserite
```

# rispetto al numero totale di risposte inserite. Nella classifica NON devono apparire i dati

# sensibili dello studente (nome, cognome, email) ma solo il codice alfanumerico.

```
DROP VIEW IF EXISTS Classifica_risposte_giusta;
```

```
CREATE VIEW
```

```
Classifica_risposte_giusta AS
```

```
SELECT
```

```
s.matricola,
```

```
(
```

```
CASE
```

```
WHEN (
```

```
SELECT
```

```
count(*) as tot_risposte
```

```
FROM
```

```
RISPOSTA r1
```

WHERE

r1.email\_studente = s.email\_studente

AND r1.TIMESTAMP IN (

SELECT

MAX(TIMESTAMP)

FROM

RISPOSTA r2

WHERE

r2.email\_studente = s.email\_studente

)

) > 0 THEN (

SELECT

COUNT(\*) as risp\_giuste

FROM

RISPOSTA r



WHERE

r.email\_studente = s.email\_studente

AND r.esito = 'GIUSTA'

AND r.TIMESTAMP IN (

SELECT

MAX(TIMESTAMP)

FROM

RISPOSTA r2

WHERE

r2.email\_studente = s.email\_studente

)

) / (

SELECT

count(\*) as tot\_risposte

FROM

```
RISPOSTA r1

WHERE

r1.email_studente = s.email_studente

AND r1.TIMESTAMP IN (

SELECT

MAX(TIMESTAMP)

FROM

RISPOSTA r2

WHERE

r2.email_studente = s.email_studente

)

)

ELSE 0

END

) AS Risposte_corrette
```

```
FROM
```

```
STUDENTE s
```

```
GROUP BY
```

```
s.matricola ,
```

```
Risposte_corrette
```

```
ORDER BY
```

```
Risposte_corrette DESC;
```

```
DELIMITER $$
```

```
CREATE PROCEDURE GetClassificaRisposteGiuste () BEGIN
```

```
SELECT
```

```
rg.matricola ,
```

```
rg.Risposte_corrette as Rapporto
```

```
FROM
```

```
Classifica_risposte_giusta as rg;
```

```
END $$ DELIMITER;
```

```
-- get classifica test completati
```

```
DELIMITER $$
```

```
CREATE PROCEDURE GetClassificaTestCompletati () BEGIN
```

```
SELECT
```

```
c.matricola ,
```

```
c.Test_conclusi
```

```
FROM
```

```
Classifica_test_completati as c;
```

```
END $$ DELIMITER;
```

```
-- stored procedure per avere l'ordine delle chiavi primarie di una tabella
```

```
DELIMITER $$
```

```
CREATE PROCEDURE IF NOT EXISTS GetPrimaryKey (IN p_table_name VARCHAR(100)) BEGIN
```

```
SELECT
```

```
ID AS INDICE ,
```

```
nome_attributo AS NOME_ATTRIBUTO,
```

```
tipo_attributo AS TIPO_ATTRIBUTO
```

```
FROM
```

```
TAB_ATT AS TA
```

```
WHERE
```

```
TA.nome_tabella = p_table_name
```

```
AND TA.key_part = "TRUE"
```

```
ORDER BY
```

```
ID ASC;
```

```
END $$ DELIMITER;
```

```
-- get tabelle delle tabelle
```

```
DELIMITER $$
```

```
CREATE PROCEDURE IF NOT EXISTS GetTabelleCreate () BEGIN
```

```
SELECT
```

```
*
```

```
FROM
```

```
TABELLA DELLE TABELLE;
```

```
END $$ DELIMITER;
```

```
-- crea GetSoluzioneQuesitoAperto
```

```
DELIMITER $$
```

```
CREATE PROCEDURE IF NOT EXISTS GetSoluzioneQuesitoAperto (IN p_id_quesito INT) BEGIN
```

```
SELECT
```

```
*  
  
FROM  
  
QUESITO_APERTO_SOLUZIONE  
  
WHERE  
  
QUESITO_APERTO_SOLUZIONE.id_quesito = p_id_quesito;  
  
  
END $$ DELIMITER;  
  
  
-- INSERT INTO TAB_ATT  
  
DELIMITER $$  
  
CREATE PROCEDURE IF NOT EXISTS InserisciAttributo (  
  
IN p_nome_tabella VARCHAR(20) ,  
  
IN p_nome_attributo VARCHAR(100),  
  
IN p_tipo_attributo VARCHAR(15)  
  
) BEGIN
```

```
INSERT INTO

TAB_ATT (nome_tabella, nome_attributo, tipo_attributo)

VALUES

(

p_nome_tabella ,

p_nome_attributo,

p_tipo_attributo

);


END $$ DELIMITER;


-- INSERT INTO TABELLA DELLE TABELLE


DELIMITER $$


CREATE PROCEDURE IF NOT EXISTS InserisciTabellaDiEsercizio (

IN p_nome_tabella VARCHAR(20),
```



```
IN p_creatore VARCHAR(100)

) BEGIN

INSERT INTO

TABELLA DELLE TABELLE (nome_tabella, creatore)

VALUES

(p_nome_tabella, p_creatore);


END $$ DELIMITER;


-- INSERT INTO VINCOLI

DELIMITER $$

CREATE PROCEDURE IF NOT EXISTS InserisciChiaveEsterna (

IN p_nome_tabella VARCHAR(20) ,

IN p_nome_attributo VARCHAR(100) ,

IN p_tabella_vincolata VARCHAR(20) ,
```

```
IN p_attributo_vincolato VARCHAR(100)

) BEGIN

-- Inserisce la chiave esterna nella tabella VINCOLI

INSERT INTO

VINCOLI (

nome_tabella ,

nome_attributo ,

tabella_vincolata ,

attributo_vincolato

)

VALUES

(

p_nome_tabella ,

p_nome_attributo ,

p_tabella_vincolata ,
```

```
p_attributo_vincolato
```

```
);
```

```
END $$ DELIMITER;
```

```
-- aggiungi chiave
```

```
DELIMITER $$
```

```
CREATE PROCEDURE IF NOT EXISTS AggiungiChiavePrimaria (
```

```
IN p_nome_tabella VARCHAR(20),
```

```
IN p_pezzo_chiave VARCHAR(100)
```

```
) BEGIN
```

```
UPDATE TAB_ATT
```

```
SET
```

```
key_part = "TRUE"
```

```
WHERE
```

```
nome_tabella = p_nome_tabella
```

```
AND nome_attributo = p_pezzo_chiave;
```

```
END $$ DELIMITER;
```

```
-- get attributi tabella
```

```
DELIMITER $$
```

```
CREATE PROCEDURE IF NOT EXISTS GetAttributiTabella (IN p_nome_tabella VARCHAR(20)) BEGIN
```

```
SELECT
```

```
nome_attributo ,
```

```
TAB_ATT.key_part AS is_key,
```

```
tipo_attributo
```

```
FROM
```

```
TAB_ATT
```

```
WHERE
```

```
nome_tabella = p_nome_tabella
```

```
ORDER BY
```

```
TAB_ATT.ID;
```

```
END $$ DELIMITER;
```

```
-- get chiavi esterne
```

```
DELIMITER $$
```

```
CREATE PROCEDURE IF NOT EXISTS GetChiaviEsterne (IN p_nome_tabella VARCHAR(20)) BEGIN
```

```
SELECT
```

```
*
```

```
FROM
```

```
VINCOLI
```

```
WHERE
```

```
nome_tabella = p_nome_tabella;
```

```
END $$ DELIMITER;
```

```
-- trigger per una tabella di prova preinserita
```

```
DELIMITER $$
```

```
CREATE TRIGGER IF NOT EXISTS after_insert_tipi_pokemon AFTER
```

```
INSERT
```

```
ON TIPI_POKEMON FOR EACH ROW BEGIN
```

```
-- Incrementa il numero di righe nella tabella
```

```
UPDATE TABELLA_DELLE_TABELLE
```

```
SET
```

```
num_righe = num_righe + 1
```

```
WHERE
```

```
nome_tabella = 'TIPI_POKEMON';
```

```
END $$ DELIMITER;

-- trigger per una tabella di prova preinserita

DELIMITER $$

CREATE TRIGGER IF NOT EXISTS after_insert_pokemon AFTER

INSERT

ON POKEMON FOR EACH ROW BEGIN

-- Incrementa il numero di righe nella tabella

UPDATE TABELLA DELLE TABELLE

SET

num_righe = num_righe + 1

WHERE

nome_tabella = 'POKEMON';

END $$ DELIMITER;
```

```
-- get ID quesito

DELIMITER $$

CREATE PROCEDURE IF NOT EXISTS GetQuesitoTest (

IN p_test_associato VARCHAR(100),

IN p_numero_quesito INT

) BEGIN

SELECT

*

FROM

QUESITO

WHERE

test_associato = p_test_associato

AND numero_quesito = p_numero_quesito;
```



```
END $$ DELIMITER;

-- inserisci quesito-tabella

DELIMITER $$

CREATE PROCEDURE IF NOT EXISTS InserisciQuesitoTabella (

IN p_id_quesito INT ,

IN p_nome_tabella VARCHAR(20)

) BEGIN

INSERT INTO

QUESITI_TABELLA (id_quesito, nome_tabella)

VALUES

(p_id_quesito, p_nome_tabella);

END $$ DELIMITER;
```

```
DELIMITER $$

CREATE PROCEDURE IF NOT EXISTS GetTabelleQuesito (IN p_test_associato VARCHAR(100)) BEGIN

SELECT DISTINCT

(nome_tabella)

FROM

QUESITI_TABELLA as QT

JOIN QUESITO as Q ON QT.id_quesito = Q.ID

WHERE

Q.test_associato = p_test_associato;

END $$ DELIMITER;

INSERT INTO

`TEST` (

`VisualizzaRisposte`,
```

```
`dataCreazione` ,  
  
`email_professore` ,  
  
`titolo`  
  
)  
  
VALUES  
  
(  
  
0 ,  
  
'2024-05-01 14:23:55',  
  
'professore@unibo.it',  
  
'Test pokemon'  
  
);  
  
  
INSERT INTO  
  
`QUESITO` (  
  
`descrizione` ,
```

```
`livello_difficolta`,
```

```
`numero_quesito` ,
```

```
`numero_risposte` ,
```

```
`test_associato` ,
```

```
`tipo_quesito`
```

```
)
```

```
VALUES
```

```
(
```

```
'Pikachu è tipo acqua',
```

```
'BASSO' ,
```

```
1 ,
```

```
0 ,
```

```
'Test pokemon' ,
```

```
'CHIUSO'
```

```
),
```

```
(  
  
  'Seleziona il NOME dei pokemon di tipo \'Electric\'',  
  
  'BASSO' ,  
  
  2 ,  
  
  0 ,  
  
  'Test pokemon' ,  
  
  'APERTO'  
  
),  
  
(  
  
  'Seleziona il nome di tutti i tipo Water oppure Fire',  
  
  'MEDIO' ,  
  
  3 ,  
  
  1 ,  
  
  'Test pokemon' ,  
  
  'APERTO'
```

```
);
```

```
INSERT INTO
```

```
`QUESITO_CHIUSO_OPZIONE` (
```

```
`id_quesito` ,
```

```
`is_corretta` ,
```

```
`numero_opzione`,
```

```
`testo`
```

```
)
```

```
VALUES
```

```
(1, 'FALSE', 1, 'Vero'),
```

```
(1, 'TRUE', 2, 'Falso');
```

```
INSERT INTO
```

```
`QUESITO_APERTO_SOLUZIONE` (
```

```
`id_quesito` ,  
  
`id_soluzione` ,  
  
`soluzione_professore`  
  
)  
  
VALUES  
  
(  
  
2 ,  
  
1 ,  
  
"SELECT nome FROM POKEMON WHERE tipo = ciao Electric ciao"  
  
) ,  
  
(  
  
2 ,  
  
2 ,  
  
'SELECT nome FROM POKEMON WHERE tipo = ciao Electric ciao'  
  
) ,
```

```
(  
  
3 ,  
  
3 ,  
  
'SELECT nome FROM POKEMON WHERE (tipo = ciao Water ciao || tipo = ciao Fire ciao ) '  
  
);
```

```
INSERT INTO
```

```
`TABELLA DELLE TABELLE` (  
  

```

```
`creatore` ,  
  

```

```
`data_creazione`,  
  

```

```
`nome_tabella` ,  
  

```

```
`num_righe`  
  

```

```
)
```

```
VALUES
```

```
(
```



```
'professore@unibo.it',
```

```
'2024-05-01 14:11:59',
```

```
'POKEMON' ,
```

```
8
```

```
),
```

```
(
```

```
'professore@unibo.it',
```

```
'2024-05-01 14:01:59',
```

```
'TIPI_POKEMON' ,
```

```
12
```

```
);
```

```
INSERT INTO
```

```
`TAB_ATT` (
```

```
`ID` ,
```

```
`key_part` ,

`nome_attributo`,

`nome_tabella` ,

`tipo_attributo`

)

VALUES

(1, 'TRUE', 'TIPO', 'TIPI_POKEMON', 'VARCHAR'),

(2, 'TRUE', 'nome', 'POKEMON', 'VARCHAR') ,

(3, 'FALSE', 'tipo', 'POKEMON', 'VARCHAR') ,

(4, 'FALSE', 'PS', 'POKEMON', 'INT') ,

(5, 'FALSE', 'is_legendary', 'POKEMON', 'INT');


INSERT INTO

`VINCOLI` (

`attributo_vincolato`,
```

```
`nome_attributo` ,  
  
`nome_tabella` ,  
  
`tabella_vincolata`  
  
)  
  
VALUES  
  
('TIPO', 'tipo', 'POKEMON', 'TIPI_POKEMON');  
  
  
  
INSERT INTO  
  
`QUESITI_TABELLA` (`ID`, `id_quesito`, `nome_tabella`)  
  
VALUES  
  
(1, 1, 'POKEMON') ,  
  
(2, 1, 'TIPI_POKEMON'),  
  
(3, 2, 'POKEMON') ,  
  
(4, 2, 'TIPI_POKEMON');
```

```
-- get matricola

DELIMITER $$

CREATE PROCEDURE IF NOT EXISTS GetMatricola (IN p_email_studente VARCHAR(100)) BEGIN

SELECT

matricola

FROM

STUDENTE

WHERE

email_studente = p_email_studente;

END $$ DELIMITER;

-- crea classifica (view) dei quesiti ordinati per numero di risposte

DROP VIEW IF EXISTS Classifica_quesitiPerNumeroRisposte;
```

```
CREATE VIEW
```

```
Classifica_quesitiPerNumeroRisposte AS
```

```
SELECT
```

```
ID ,
```

```
test_associato,
```

```
numero_quesito,
```

```
numero_risposte
```

```
FROM
```

```
QUESITO
```

```
ORDER BY
```

```
numero_risposte DESC;
```

```
-- get classifica quesiti
```

```
DELIMITER $$
```

```
CREATE PROCEDURE IF NOT EXISTS GetClassificaQuesitiPerNumeroRisposte () BEGIN
```

```
SELECT

test_associato,

numero_quesito,

numero_risposte

FROM

Classifica_quesitiPerNumeroRisposte;


END $$ DELIMITER;


-- elimina tabella

DELIMITER $$

CREATE PROCEDURE IF NOT EXISTS EliminaTabella (IN p_nome_tabella VARCHAR(20)) BEGIN

DELETE FROM TABELLA DELLE TABELLE

WHERE

nome_tabella = p_nome_tabella;
```

```
END $$ DELIMITER;
```

```
-- GetQuesitiAssociatiAlTest
```

```
DELIMITER $$
```

```
CREATE PROCEDURE IF NOT EXISTS GetQuesitiAssociatiAlTest (IN p_test_associato VARCHAR(100)) BEGIN
```

```
SELECT
```

```
*
```

```
FROM
```

```
QUESITO
```

```
WHERE
```

```
test_associato = p_test_associato
```

```
ORDER BY
```

```
numero_quesito ASC;
```

```
END $$ DELIMITER;

-- GetTabelleQuesitiNum

DELIMITER $$

CREATE PROCEDURE IF NOT EXISTS GetTabelleQuesitiNum (IN p_id_quesito INT) BEGIN

SELECT

nome_tabella

FROM

QUESITI_TABELLA as QT

WHERE

QT.id_quesito = p_id_quesito;

END $$ DELIMITER;

-- get test
```



```
DELIMITER $$
```

```
CREATE PROCEDURE IF NOT EXISTS GetTest (IN p_titolo VARCHAR(100)) BEGIN
```

```
SELECT
```

```
*
```

```
FROM
```

```
TEST
```

```
WHERE
```

```
titolo = p_titolo;
```

```
END $$ DELIMITER;
```

```
-- get professori
```

```
DELIMITER $$
```

```
CREATE PROCEDURE IF NOT EXISTS GetProfessori () BEGIN
```

```
SELECT
```

```
email_professore

FROM

PROFESSORE;

END $$ DELIMITER;

DELIMITER $$

CREATE PROCEDURE IF NOT EXISTS CheckRisultatiStudente (IN p_email_studente VARCHAR(100)) BEGIN

SELECT

COUNT(*) as 'check'

FROM

SVOLGIMENTO_TEST st

where

st.email_studente = p_email_studente

and stato = "CONCLUSO";
```

```
END $$ DELIMITER;
```

```
-- get GetTestDelloStudente
```

```
DELIMITER $$
```

```
CREATE PROCEDURE IF NOT EXISTS GetTestDelloStudente (IN p_email_studente VARCHAR(100)) BEGIN
```

```
SELECT
```

```
*
```

```
FROM
```

```
SVOLGIMENTO_TEST
```

```
JOIN TEST ON SVOLGIMENTO_TEST.titolo_test = TEST.titolo
```

```
WHERE
```

```
email_studente = p_email_studente;
```

```
END $$ DELIMITER;
```

```
-- cerca utente
```

```
DELIMITER $$
```

```
CREATE PROCEDURE IF NOT EXISTS CercaUtente (IN p_email VARCHAR(100)) BEGIN
```

```
SELECT
```

```
email
```

```
FROM
```

```
UTENTE
```

```
WHERE
```

```
email = p_email;
```

```
END $$ DELIMITER;
```

```
-- GetTabelleRiferite
```

```
DELIMITER $$
```

```
CREATE PROCEDURE IF NOT EXISTS GetTabelleRiferite (IN p_nome_tabella VARCHAR(20)) BEGIN
```

```
SELECT DISTINCT
```

```
(tabella_vincolata)
```

```
FROM
```

```
VINCOLI
```

```
WHERE
```

```
nome_tabella = p_nome_tabella;
```

```
END $$ DELIMITER;
```

```
-- GetTestsDelProfessoreAperti
```

```
DELIMITER $$
```

```
CREATE PROCEDURE IF NOT EXISTS GetTestsDelProfessoreAperti (IN p_email_professore VARCHAR(100)) BEGIN
```

```
SELECT
```

```
*
```

FROM

TEST

WHERE

email\_professore = p\_email\_professore

AND VisualizzaRisposte = 0;

END \$\$ DELIMITER;

-- getInfoProfessore

DELIMITER \$\$

CREATE PROCEDURE IF NOT EXISTS GetInfoProfessore (IN p\_email\_professore VARCHAR(100)) BEGIN

SELECT

\*

FROM

PROFESSORE

```
JOIN UTENTE ON PROFESSORE.email_professore = UTENTE.email
```

```
WHERE
```

```
email_professore = p_email_professore;
```

```
END $$ DELIMITER;
```

```
-- GetInfoStudente
```

```
DELIMITER $$
```

```
CREATE PROCEDURE IF NOT EXISTS GetInfoStudente (IN p_email_studente VARCHAR(100)) BEGIN
```

```
SELECT
```

```
*
```

```
FROM
```

```
STUDENTE
```

```
JOIN UTENTE ON STUDENTE.email_studente = UTENTE.email
```

```
WHERE
```

```
email_studiante = p_email_studiante;
```

```
END $$ DELIMITER;
```

```
-- EliminaQuesito
```

```
DELIMITER $$
```

```
CREATE PROCEDURE IF NOT EXISTS EliminaQuesito (IN p_id_quesito INT) BEGIN
```

```
DELETE FROM QUESITO
```

```
WHERE
```

```
ID = p_id_quesito;
```

```
END $$ DELIMITER;
```

```
-- EliminaTest
```

```
DELIMITER $$
```



```
CREATE PROCEDURE IF NOT EXISTS EliminaTest (IN p_titolo VARCHAR(100)) BEGIN
```

```
DELETE FROM TEST
```

```
WHERE
```

```
titolo = p_titolo;
```

```
END $$ DELIMITER;
```