RELAZIONE PROGETTO BASI DI DATI

Traccia: Gestione Esami Universitari

892721 - Eleonora Greco
892075 - Domenico Sosta
892604 - Donald Gera

Università
Ca'Foscari
Venezia

Indice

1 Introduzione **2**

2 Progettazione della base di dati **2**
2.1 Raccolta ed analisi dei requisiti . . . 2
2.2 Funzionalità principali . . . 3
2.3 Ruoli . . . 6
2.4 Progettazione concettuale della base di dati . . . 7
2.4.1 Schema ad oggetti . . . 7
2.4.2 Giustificazione delle scelte progettuali . . . 8
2.5 Progettazione logica della base di dati . . . 9
2.5.1 Schema logico . . . 9
2.5.2 Giustificazione delle scelte progettuali . . . 11
2.6 Integrità dei dati: vincoli e trigger . . . 12
2.7 Indici . . . 19

3 Sviluppo back-end **21**
3.1 Scelta di Flask come framework . . . 21
3.1.1 Familiarità e competenza . . . 21
3.1.2 Adattabilità e flessibilità . . . 21
3.1.3 Funzionalità offerte da Flask . . . 21
3.2 Struttura dell'applicazione . . . 21
3.2.1 Struttura della directory e dei moduli . . . 22
3.2.2 Componenti chiave . . . 22
3.3 Sicurezza . . . 23
3.3.1 Autenticazione e Autorizzazione . . . 23
3.3.2 Decorator . . . 24
3.3.3 Protezione da SQL Injection e XSS . . . 26
3.4 Interazione con il DBMS . . . 26
3.4.1 Utilizzo di SQLAlchemy . . . 26
3.4.2 Query principali e gestione mediante metodi . . . 27

4 Implementazione front-end **30**
4.1 Tecnologie utilizzate . . . 30
4.2 Scelte di design . . . 30
4.3 Integrazione tra front-end e back-end . . . 30

5 Contributo al progetto **33**
5.1 Suddivisione compiti . . . 33
5.2 Piattaforme di collaborazione . . . 33

1 Introduzione

Il presente documento costituisce un'analisi approfondita del progetto, dedicando particolare attenzione alle funzionalità implementate e agli aspetti progettuali e tecnici sviluppati. L'obiettivo principale di questo progetto è la creazione di un'applicazione web volta a consentire ai docenti universitari di gestire e registrare le valutazioni degli studenti relativamente a specifici corsi. Questo sistema include funzionalità avanzate, tra cui permettere l'assegnamento di punteggi bonus, specificare la propedeuticità di un esame, permettere la gestione di diverse prove d'esame a docenti differenti, nonchè fornire vari storici delle prove sostenute da un determinato studente. L'analisi del progetto si estende a esaminare dettagliatamente l'implementazione di queste caratteristiche, fornendo una visione chiara degli aspetti che ne regolano la progettazione e lo sviluppo.

2 Progettazione della base di dati

L'efficacia dell'applicazione web realizzata, dedicata alla gestione degli esami, dipende in larga misura dalla qualità e dall'efficienza della sua base di dati. In questa sezione esploreremo dettagliatamente la progettazione e la modellazione della base di dati attraverso diverse fasi del processo che includono: l'identificazione e la comprensione delle esigenze del sistema con conseguente analisi dei requisiti raccolti. Questo primo passo permette in seguito di definire le entità, le relazioni e gli attributi chiave necessari per soddisfare le esigenze del sistema. Seguono, in base alle informazioni raccolte nella fase di analisi, la progettazione concettuale e logica.
In conclusione, affrontiamo aspetti critici come l'integrità dei dati, essenziale per assicurare la coerenza e l'affidabilità delle informazioni, il controllo degli accessi volto a proteggere dati sensibili, e le considerazioni sulle prestazioni del sistema. Quest'ultime includono l'analisi delle query più frequenti e la successiva creazione di indici per ottimizzare l'efficienza di tali interrogazioni e garantire una risposta tempestiva alle richieste degli utenti.

2.1 Raccolta ed analisi dei requisiti

La fase di raccolta ed analisi dei requisiti è risultata fondamentale per comprendere e definire le specifiche del sistema. Durante questo processo iniziale, abbiamo identificato con precisione le esigenze degli utenti e le funzionalità che la base di dati deve supportare al fine di assicurare il corretto funzionamento dell'intero sistema.
Il primo passo è stato a partire dalla consegna derivare le entità principali. Come precedentemente menzionato, l'applicazione che è stata sviluppata è finalizzata a fornire una piattaforma per la gestione degli esami da parte dei docenti e studenti. Il sistema deve essere in grado di gestire la creazione di esami ognuno associato ad un set di prove, lo svolgimento delle suddette prove in determinati appelli, la registrazione dei voti e infine fornire supporto per l'autenticazione e l'autorizzazione degli utenti. Le entità chiave che sono state individuate a partire della consegna sono:

- **Esami:** contiene informazioni riguardanti gli esami, come il nome, il numero di CFU ottenuti alla formalizzazione di un esame, che rappresentano un indicatore di ore requisite a sostenere con successo l'esame in questione, e l'anno di svolgimento. Un esame si può formalizzare solamente quando tutte le prove ad esso associato sono state superate e sono valide.
  È importante sottolineare che, dal punto di vista dell'applicazione, trattiamo gli esami come corsi. I docenti non li creano direttamente, ma è la figura di admin ad assegnare un corso a un docente. Gli insegnanti si occupano invece di creare le prove, personalizzandole secondo le proprie preferenze. Il motivo di questa scelta è, in primo luogo, la volontà di mantenere una netta divisione dei compiti dei docenti. Questi hanno la competenza di modificare solo prove da loro create, ovviamente rimane la possibilità di visionare l'esame completo.
  Nel caso contrario, se consentissimo a un singolo insegnante di creare un esame gestito da più docenti, dovremmo permettere anche eventuali aggiornamenti per far fronte a possibili errori commessi. Ciò si traduce nel bisogno di una figura di insegnante di riferimento per ogni esame con la possibilità di modificare tutto, anche ciò che non rientra nelle sue competenze. In alternativa si potrebbe non abilitare modifiche o cancellazioni, ma renderebbe l'applicativo rigido, suscettibile agli errori umani.
  In secondo luogo, la procedura risulta semplificata dal punto di vista del docente. Non

è suddivisa in due step, prima creare un esame generale, e poi crearne una frazione, ma richiede semplicemente la definizione di una prova. Per questi motivi, abbiamo concluso fosse più appetibile lasciare ai docenti solo la gestione delle prove, senza creare la figura di docente referente che avrebbe complicato ulteriormente il nostro database.

- **Prove:** serve a rappresentare le prove associate ai vari esami che possono essere sostenute in determinati **Appelli**. In particolare, vengono memorizzate informazioni quali la tipologia (scritto, orale, progetto) e il peso relativo alla prova che serve ad indicare la diversa ricaduta sul risultato finale e superamento dell'esame. Possiamo infatti avere prove di idoneità (che non influenzano il risultato finale ma che vanno superate per avere l'esame formalizzabile), prove il cui voto contribuisce al voto finale relativo all'esame e prove non obbligatorie ma che forniscono un bonus alla valutazione finale. Nel caso di prove di idoneità o che forniscono bonus il peso sarà 0.0 in quanto esse non contribuiranno al calcolo della media pesata. Per quanto riguarda le altre prove il voto contribuirà alla media pesata del voto finale e il loro peso sarà arbitrario e scelto dal docente in fase di definizione delle diverse prove. E' rilevante sottolineare il fatto che le diverse prove d'esame potrebbero richiedere a loro volta il superamento di altre prove per poter essere sostenute, ad esempio una prova Prova2 potrebbe dover richiedere il superamento di Prova1 per poter essere sostenuta.

- **Utenti:** rappresenta gli utenti del sistema e contiene informazioni quali nome, cognome, password ed email. Gli utenti si suddividono in

  - **Docenti:** rappresenta un insegnante universitario autorizzato a gestire esami e a creare prove. Il docente è responsabile della configurazione delle prove, della valutazione e della gestione degli esami. Le diverse prove relative ad uno stesso esame possono essere gestite da docenti diversi, in questo caso ogni docente ha potere soltanto sulle prove da esso creato e non può interferire con le prove degli altri docenti.
  - **Studenti:** identifica uno studente universitario con le relative informazioni associate, matricola e recapito telefonico. Gli studenti possono sostenere prove in determinati **Appelli** e ricevere valutazioni relative agli appelli così da poter superare gli esami e formalizzarli quando tutte le prove relative all'esame da formalizzare devono essere superate e valide.
  - **Admin:** l'account amministratore (Admin) assume il ruolo di utente con responsabilità di segreteria all'interno del sistema. La sua principale incombenza consiste nella creazione di nuovi profili per studenti e docenti, includendo tutti i relativi attributi necessari. Inoltre, l'Admin ha il compito di inserire nel database le informazioni sugli esami disponibili e di assegnarli ai docenti competenti per la gestione.

- **Appelli:** rappresenta un appello durante il quale gli studenti possono sostenere una prova. Contiene informazioni quali data e luogo in cui avrà svolgimento una prova.

Queste entità costituiscono la struttura di base per la progettazione del database.

2.2 Funzionalità principali

- Il sistema permette l'aggiunta all'applicativo in modo semplice e agevole di studenti, docenti e di esami andando a definirne tutte le caratteristiche necessarie, e permette l'associazione di un determinato esame ad uno o più insegnanti.

- **Gestione di Esami:** I docenti possono gestire gli esami, definendo le prove richieste per il superamento.

- **Definizione delle prove:** I docenti hanno la facoltà di creare e definire le prove associate a un esame. Ogni prova è caratterizzata da diverse specifiche, tra cui il suo nome, la tipologia (scritto, orale o progetto), il peso relativo alla valutazione finale dell'esame, il punteggio massimo ottenibile ed il punteggio minimo richiesto per il superamento della prova, che vengono settati rispettivamente di default a 31 e 18, e le prove pre-requisite che devono essere superate prima di affrontare quella specifica prova. Inoltre, è possibile stabilire se una prova è considerata d'idoneità, ossia se non influisce sulla valutazione finale dell'esame ma è comunque necessaria per completarlo, oppure se si tratta di una prova non obbligatoria

che non concorre al calcolo della media pesata del risultato finale, ma che fornisce un bonus aggiuntivo. Infine, dal momento che prove relative ad un medesimo esame potrebbero essere gestite da docenti diversi allora tutti i docenti associati ad un esame specifico avranno accesso completo alle informazioni relative all'esame, anche se inserite da altri docenti. Tuttavia, saranno autorizzati a modificare solamente le entità che essi hanno creato.

- **Creazione ed eliminazione appelli:** Un docente può creare appelli definendo data e luogo e può eliminare appelli non ancora svolti.
- **Iscrizione agli appelli:** Gli studenti possono sostenere prove agli appelli. Ogni appello ha una data di svolgimento e quando una prova viene superata viene assegnata anche una data di scadenza. Trascorsa tale scadenza, la prova non è più considerata valida nel sistema e viene trattata come se non fosse stata superata. Un'altra circostanza che porta all'invalidazione di una prova è il suo sostenimento (sia in caso di superamento sia in caso di fallimento) in un appello successivo nel caso in cui uno studente avesse già superato la prova. Gli studenti inoltre possono cancellare la loro iscrizione dagli appelli fino al giorno di svolgimento dell'appello.
- **Formalizzazione esami:** Gli studenti possono formalizzare i vari esami quando tutte le prove necessarie sono state superate e sono ancora valide.
- **Valutazione appelli:** I docenti possono inserire i voti agli studenti una volta che l'appello relativo è stato svolto.
- **Visualizzazione dello stato degli appelli:** I docenti possono visualizzare lo stato degli appelli, ovvero nel caso in cui l'appello non si sia ancora verificato possono visualizzare l'elenco degli studenti iscritti all'appello mentre nel caso in cui l'appello si sia già svolto i docenti possono anche visualizzare le informazioni degli studenti che hanno sostenuto la prova (con voto relativo se questo è stato già assegnato).
- **Elenco degli studenti per cui è possibile la formalizzazione di un esame:** Un docente è in grado di visionare, relativamente ad un esame che gestiscono, tutti gli studenti che hanno i requisiti necessari per il superamento di tale esame e sono in grado di formalizzarlo ma non lo hanno ancora fatto.
- **Visualizzazione dello stato degli studenti:** Il sistema fornisce agli studenti la visualizzazione del proprio stato accademico, con visualizzazione di prove valide superate e storico delle prove sostenute con relativi voti.
- **Visualizzazione del piano di studi:** Ogni studente ha la possibilità di vedere il suo avanzamento del percorso universitario monitorando gli esami che deve svolgere, i risultati che ottiene per ogni esame e la media aritmetica e ponderata dei voti fino al momento di visualizzazione.
- **Calcolo del risultato dell'esame:** Viene automatizzato il calcolo della valutazione per ogni esame. Dato uno studente quest'ultima viene calcolata nel seguente modo:
  - si considerano tutte le prove obbligatorie (prove classiche e di idoneità) relative all'esame
  - si controlla se tutte queste sono state superate e sono valide (ovvero relative all'ultimo appello)
  - si esegue una media pesata dei voti delle prove
  - si sommano alla media pesata eventuali bonus accumulati mediante il superamento di prove non obbligatorie che forniscono punti aggiuntivi

  Una volta eseguito il calcolo, che è automatico ed eseguito mediante un trigger, non viene consentito al docente di intervenire manualmente per modificare la valutazione sia nel caso l'esame non risulti ancora non formalizzato sia una volta registrato.
- **Visualizzazione dei docenti relativi ad un esame:** Questa funzionalità permette agli utenti di esaminare l'elenco completo dei docenti associati a uno specifico esame. Vengono forniti dettagli quali nome e cognome dei docenti coinvolti nel processo di valutazione dell'esame.

- **Ricerca dei docenti da aggiungere ad un esame:** Si consente ai docenti relativi ad un esame di effettuare una ricerca mirata tra i docenti disponibili, al fine di identificare quelli adatti da aggiungere a un determinato esame.
- **Visualizzazione dettagli appelli:** Questa funzione offre agli utenti la possibilità di esplorare in modo dettagliato le informazioni relative agli appelli. Ciò include date, luoghi, eventuali variazioni o aggiornamenti riguardanti gli appelli associati a un esame specifico. Gli utenti possono ottenere una visione completa degli appelli in corso o futuri, nonché dei dettagli relativi agli appelli passati.
- **Logout:** La funzionalità di logout consente agli utenti di terminare la loro sessione in modo sicuro e di disconnettersi dall'applicazione. Ciò garantisce la protezione delle informazioni personali e mantiene la sicurezza dell'account. Al momento del logout, gli utenti vengono reindirizzati alla pagina di accesso, dove possono successivamente effettuare il login per accedere nuovamente all'applicazione.

Qui sono presentati solamente i vincoli preliminari per le varie funzionalità; per una trattazione dettagliata dei vincoli associati a specifiche operazioni, consultare la sezione Integrità dei dati: vincoli e trigger.

2.3 Ruoli

In modo da permettere una gestione efficace dell'applicazione e contribuire alla sicurezza complessiva del sistema è cruciale identificare chiaramente i ruoli e le politiche di autorizzazione al fine di garantire la sicurezza e una corretta separazione delle responsabilità all'interno dell'applicazione web. La definizione di ruoli specifici è essenziale per assicurare che gli utenti abbiano accesso solo alle informazioni rilevanti alle loro funzioni e che siano in grado di svolgere solo le attività permesse. Nel contesto della nostra applicazione possiamo individuare quattro ruoli principali Admin, Utente, Studente, Docente dove per ognuno nella seguente tabella sono indicate le varie operazioni concesse sulle diverse tabelle del database.

Tabella	**Admin**	**Utente**	**Docente**	**Studente**
Esami	ALL	—	SELECT	SELECT
Prove		—	SELECT, INSERT, DELETE	SELECT
Appelli		—	SELECT, INSERT, UPDATE, DELETE	SELECT
Studenti		SELECT	SELECT	SELECT
Docenti		SELECT	SELECT	SELECT
Iscrizioni		—	SELECT, UPDATE	SELECT, INSERT, DELETE
FormalizzazioneEsami		—	SELECT, UPDATE	SELECT, UPDATE
Superamento		—	SELECT, INSERT, DELETE	SELECT

- **Utente:** Ruolo di base con accesso limitato e senza privilegi specifici. Consente l'accesso all'applicazione e la visualizzazione delle informazioni di base. Serve ai fini dell'autenticazione ed una volta autenticato l'utente il DBMS sa con chi sta interagendo e dunque assegna il ruolo specifico. In sintesi dunque, fornisce una base di accesso minima per garantire sicurezza e la privacy dei dati ed in seguito permettere l'accesso alle varie funzionalità dell'applicativo.
- **Admin:** L'assegnazione completa di privilegi al ruolo di amministratore è stata concepita con l'obiettivo di conferire un controllo totale sulla base di dati. Questa scelta è stata adottata per consentire all'amministratore di intervenire prontamente in situazioni in cui potrebbero verificarsi problemi a causa di un utilizzo scorretto del sistema o di piccoli bug non ancora individuati. Concedere tutti i privilegi all'amministratore significa garantire un controllo esteso sulla base di dati e ciò consente all'amministratore di eseguire operazioni di manutenzione, correzione e ottimizzazione in qualsiasi parte del sistema.
- **Docente:** Ruolo dedicato ai docenti. Ha la possibilità di creare, modificare e cancellare prove associate agli esami che gestisce. Può visualizzare lo stato di studenti (prove valide sostenute, storico delle prove sostenute), appelli (studenti che hanno superato le diverse prove) e gli esami. Inoltre è in grado di vedere l'elenco degli studenti che sono in condizione di avere l'esame registrato (ma non lo hanno ancora fatto) e le relative caratteristiche delle prove che abilitano la registrazione dell'esame. Le operazioni consentite e i relativi vincoli sono trattati in dettaglio nella sezione Integrità dei dati: vincoli e trigger.
- **Studente:** Ruolo per gli studenti universitari. Può iscriversi agli appelli, sostenere le prove, visualizzare i risultati relativi e formalizzare gli esami nel caso in cui sia nelle condizioni di poterlo fare. Le operazioni consentite e i relativi vincoli sono trattati in dettaglio nella sezione Integrità dei dati: vincoli e trigger.

L'implementazione dei ruoli è stata guidata dal principio del privilegio minimo. Questo approccio sottolinea la necessità di concedere agli utenti solo le operazioni strettamente necessarie per svolgere le proprie funzioni, riducendo al minimo il rischio di abusi, accessi non autorizzati e garantendo una gestione più sicura e controllata del sistema dal momento che si organizzano in modo efficiente le responsabilità all'interno del sistema.

2.4 Progettazione concettuale della base di dati

Dopo l'analisi e la raccolta dei requisiti, il secondo fondamento essenziale per la progettazione e modellazione del nostro database è rappresentato dalla progettazione concettuale. Tale fase consente di strutturare gli elementi chiave del sistema precedentemente identificati. Nel nostro contesto, abbiamo adottato Lucidchart come strumento per sviluppare un modello ad oggetti che delinei le tabelle, le relazioni, i vincoli e le invarianti, fornendo una visione comprensiva dell'architettura del database. Nel caso in cui la visualizzazione dello schema risulti ostacolata a causa delle dimensioni, è disponibile la visione mediante il seguente link.

2.4.1 Schema ad oggetti

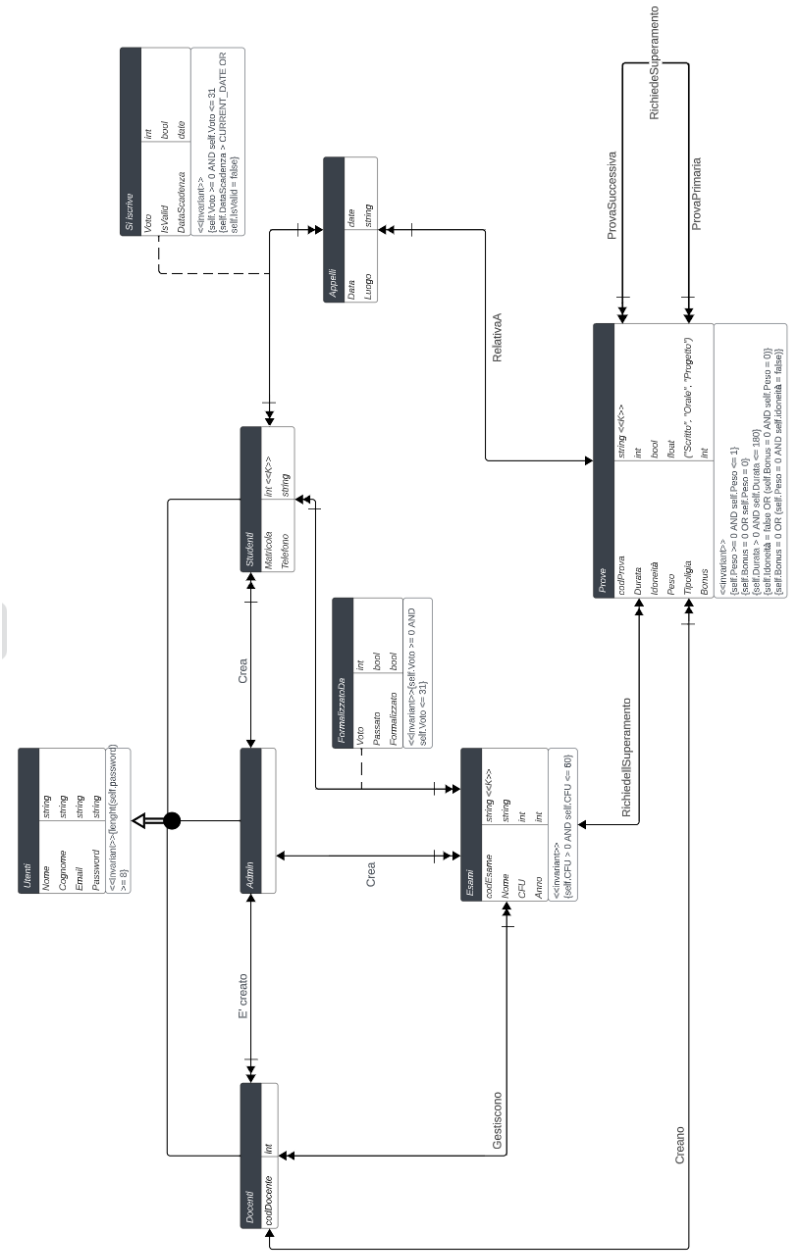

Figura 1: Schema ad oggetti

2.4.2 Giustificazione delle scelte progettuali

Segue una spiegazione delle principali decisioni di progettazione adottate per assicurare la scalabilità, l'efficienza e la facilità di manutenzione del sistema:

- È stata creata una gerarchia di tipi a partire dalla classe Utente al fine di evitare duplicazioni di informazioni. Le tre sottoclassi, Admin, Studenti e Docenti sono delle partizioni, cioè detengono sia il vincolo di disgiunzione che di copertura. Questa scelta agevola la corretta implementazione anche durante la fase di progettazione logica.

- La gestione del vincolo relativo al superamento di prove propedeutiche sulle prove d'esame è effettuata tramite la relazione molti a molti ricorsiva (richiedeSuperamento) in Prove. Così facendo indichiamo il fatto che una prova può richiedere il superamento di più prove per essere sostenuta e a sua volta può essere richiesta come prova da superare da più prove.

- Vincoli relativi alla lunghezza delle password per le tre categorie di utenti: essenziali per assicurare che le password degli utenti siano lunghe almeno 8 caratteri. Tale requisito accresce la sicurezza degli account, diminuendo il rischio di accessi non autorizzati mediante attacchi brute-force o altre tecniche di hacking. Sebbene tale controllo possa essere effettuato anche lato back-end, implementarlo direttamente nel database consente di centralizzare il vincolo, facilitando così la gestione e il mantenimento.

- Tra i diversi vincoli preliminari, quelli più significativi sono relativi alla tabella delle Prove:

  – self.Idoneità = false OR (self.Bonus = 0 AND self.Peso = 0): Con questo vincolo intendiamo specificare che se una prova è di idoneità, allora il suo peso e bonus devono essere entrambi pari a 0. Questo è logicamente equivalente a dire che la prova non è di idoneità, oppure il suo bonus e peso sono entrambi 0.

  – self.Bonus = 0 OR (self.Peso = 0 AND self.idoneità = false): Questo vincolo esprime il concetto che se una prova fornisce un bonus, il suo peso deve essere 0 e non può essere una prova di idoneità (idoneità = false). Questo è logicamente equivalente a dire che la prova non è una prova bonus, oppure il suo peso è 0 e non è di idoneità.

- Abbiamo optato per l'uso di un tipo enumerato denominato "tipologia" al fine di rappresentare lo stato delle prove, gestendo in modo specifico le diverse categorie (scritto, orale, progetto). Questo approccio assicura che solo valori predefiniti possano essere inseriti nella colonna "tipologia" della tabella delle prove, garantendo così la coerenza dei dati e semplificando le eventuali interrogazioni al database.

- Sono presenti delle parzialità tra le varie tabelle, di seguito motiviamo quelle interessanti:

  – **Docenti - Prove:** Un docente potrebbe non aver ancora stabilito le modalità d'esame, quindi non avere ancora nessuna prova a suo carico. Tuttavia una prova viene creata sicuramente da un docente, da questo lato l'associazione è totale.

  – **Appelli - Prove:** In un dato momento potrebbe non essere stato creato nessun appello per una prova, ma un appello si riferisce sempre a una prova quindi da questo lato l'associazione è totale.

  – **Prove - Prove:** Una prova può richiedere il superamento di una, nessuna o più prove.

  – **Esami - Studenti:** La parzialità riguarda entrambi i versi in quanto uno studente può non aver sostenuto nessun esame e un esame non essere mai stato sostenuto da uno studente, ad esempio se l'esame è recente.

  – **Studenti - Appelli:** La parzialità riguarda entrambi i versi.

  – **Docenti - Esami:** Al momento dell'inserimento di un nuovo docente nel database questo non ha alcun esame associato, quindi è presente la parzialità. Invece, dal lato opposto, un esame viene sicuramente assegnato ad almeno un insegnate, quindi la relazione è totale

  – **Admin:** Tutte le relazioni con la tabella Admin hanno una parzialità in quanto può non creare nessun Docente, Esame o Studente.

2.5 Progettazione logica della base di dati

2.5.1 Schema logico

Si fornisce lo schema logico, presentato sia dal punto di vista grafico (includendo i tipi degli attributi) che testuale. Nella rappresentazione testuale, le chiavi primarie sono sottolineate per una chiara identificazione, mentre le chiavi esterne sono esplicitamente indicate con un asterisco (*). Per la rappresentazione grafica, nel caso le dimensioni ne impediscano una chiara lettura, si rimanda al seguente link.

Versione Testuale:

- **Studenti** (Nome, Cognome, Email, Password, Telefono, Matricola)
- **Formalizzazione Esami** (Voto, Passato, Formalizzato, Studente*, Esame*)
  - Studente FK(Studenti)
  - Esame FK(Esami)
- **Esami** (CodEsami, Nome CFU, Anno)
- **Gestiscono** (Esame*, Docente*)
  - Esame FK(Esami)
  - Docente FK(Docenti)
- **Docenti** (Nome, Cognome, Email, Password, CodDocente)
- **Prove** (Bonus, Idoneità, Peso, Esame*, CodProve, Docente*, Tipologia, Durata)
  - Prova FK(Prove)
  - Docente FK(Docenti)
- **Superamenti** (ProvaSuccessiva*, ProvaPrimaria*)
  - ProvaSuccessiva FK(prove)
  - ProvaPrimaria FK(prove)
- **Appelli** (CodAppelli, Data, Luogo, Prova*)
  - Prova FK(Prove)
- **Iscrizioni** (Voto, IsValid, Studente*, Appello*, DataScadenza)
  - Studente FK(Studenti)
  - Appello FK(Appelli)

Versione grafica:

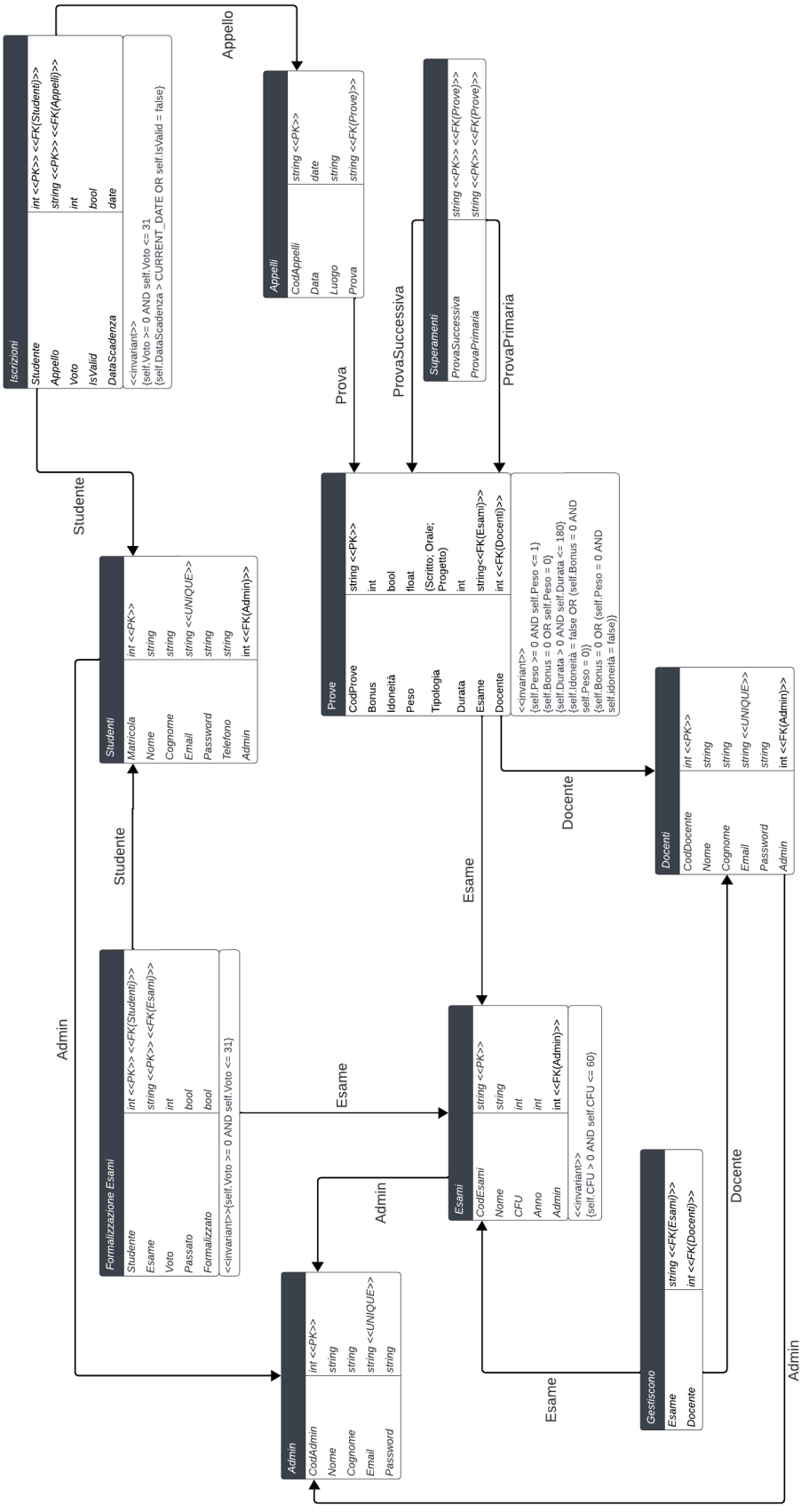

Figura 2: Schema Logico

2.5.2 Giustificazione delle scelte progettuali

Per tradurre da schema concettuale ad oggetti a logico relazionale ci siamo attenuti rigorosamente alle regole mostrate a lezione. In particolare:

- **Le sottoclassi Studenti e Insegnanti:** Traduzione tramite partizionamento orizzontale in quanto la classe utenti non ha alcuna associazione con altre classi, oltre che pochi attributi. Inoltre è poco interessante avere una visione d'insieme di tutti gli utenti e anzi si preferisce avere due relazioni distinte. Le altre due opzioni sarebbero risultate poco adatte, ovvero con una relazione unica avremmo troppi valori nulli in un'unica tabella mentre il partizionamento verticale complicherebbe il recupero di tutte le informazioni relative ad un'entità.
- **Attributo email:** Traduzione con l'aggiunta del vincolo UNIQUE per permettere che ogni email, sia per studenti che per docenti, sia effettivamente univoca ma non chiave primaria (ruolo coperto dalla matricola o dal codDocente)
- **Chiavi primarie:** Aggiunto un attributo di identificazione univoco per tutte le classi. Per quelle che non avevano una chiave nello schema a oggetti è stata creata una chiave sintetica di tipo stringa identificabile come cod+"Nome tabella". Notare che il codice docente era invece una chiave già presente, per questa ragione non segue questo standard ed è di tipo intero
- **Associazione ricorsiva:** L'associazione ricorsiva della relazione "Prove" è stata tradotta con una nuova relazione che contiene due chiavi esterne che riferiscono la relazione "Prove" in quanto l'associazione è (N:N) e quindi risulta essere l'opzione formalmente più corretta.
- **Associazioni molti a molti:** Le associazioni (N:N) sono state tradotte aggiungendo allo schema tabelle intermedie, quali Iscrizioni e FormalizzazioneEsami. Queste tabelle agevolano l'associazione tra studenti e appelli, nonché tra studenti ed esami, garantendo una maggiore flessibilità del sistema e semplificando le interrogazioni tra tabelle correlate. Esse contengono due chiavi esterne, nonchè primarie, che riferiscono le due tabelle principali. Vengono qui inseriti anche gli attributi dell'associazione, se presenti.
- **Associazioni molti a uno:** Le associazioni (N:1)/(1:N) sono state tradotte aggiungendo agli attributi della relazione univoca una chiave esterna riferisce l'altra relazione. Nel nostro schema non sono presenti associazioni (N:1) che nella parte di unicità sono anche parziali, quindi non abbiamo alternative per la traduzione.
- **Note:** Nel nostro schema non sono presenti associazioni uno a uno, attributi composti e attributi multivalore.

2.6 Integrità dei dati: vincoli e trigger

Al fine di mantenere la coerenza della base di dati e semplificare alcune operazioni, sono stati realizzati diversi trigger. Questi trigger sono progettati per impedire, automatizzare e facilitare determinate azioni all'interno del sistema, contribuendo così a garantire una gestione efficiente e coerente dei dati. Di seguito vengono presentate le politiche di integrità nei confronti delle operazioni consentite su diverse tabelle e nei casi più interessanti vengono forniti i trigger nel caso di impedimento di una determinata operazione se questa rompe il vincolo imposto o se in seguito ad un'operazione ne segue automaticamente un'altra.
Dal momento che le entità su cui i vari utenti possono effettuare operazioni sono Prove, Appelli, Superamenti, Iscrizioni, FormalizzazioneEsami ed Esami, per ognuna di queste tabelle verranno considerate le varie operazioni eseguibili su di esse e per ciascuna operazione i vincoli relativi.

APPELLI

- **Delete:** Consentite su appelli che non si sono ancora verificati (ovvero tali che Appelli.data < CurrentDate)
- **Update:** Consentiti gli aggiornamenti su data e luogo dell'appello se l'appello non si è verificato
- **Insert:**
  - Nel momento di creazione di un appello relativo a un esame, la somma totale dei pesi delle prove associate all'esame deve essere 1 a meno che l'intero set di prove che compongono l'esame sia di idoneità. In caso contrario, la creazione dell'appello viene impedita, e la responsabilità di gestire eventuali prove e riorganizzare il corso viene affidata ai professori. È importante sottolineare che, al momento della creazione di un appello, l'esame e le relative prove sono considerati attentamente pianificati dai docenti. Di conseguenza, non saranno possibili ulteriori modifiche relative all'esame e alle prove corrispondenti dopo l'avvenimento di un appello relativo ad una di tali prove. La politica di gestione delle prove verrà tratta in dettaglio nella sezione relativa.

```
CREATE TRIGGER check_if_sum_equals_1
BEFORE INSERT ON Appelli
FOR EACH ROW EXECUTE FUNCTION check_if_sum_equals_1_function();

CREATE OR REPLACE FUNCTION check_if_sum_equals_1_function()
RETURNS TRIGGER AS $$
BEGIN
    IF(1 !=  SELECT SUM(peso) FROM Prove WHERE esame IN
             (SELECT esame FROM Prove WHERE NEW.prova = codProva;);)
             THEN RETURN NULL;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Trigger 1: controlla che il peso totale delle prove che costituiscono un esame nel momento di creazione di un appello per una prova del suddetto esame sia 1

- Un docente può creare l'appello per una prova che richiede il superamento di altre prove anche se non esistono appelli per queste seconde prove, è responsabilità dei docenti assicurarsi che sia possibile svolgere le prove secondo l'ordine stabilito al momento della definizione delle prove. La conseguenza di una svista di un docente è semplicemente che gli studenti non si potranno iscrivere a determinati appelli.

PROVE e SUPERAMENTI

- **All:** Dal momento che un corso è gestito da più docenti, consentiamo loro la creazione di un numero arbitrario di prove, il peso totale delle prove esistenti per un esame può essere qualsiasi ma come abbiamo specificato precedentemente affinché un docente possa creare un appello il peso totale di tutte le prove relative all'esame associato alla prova per cui si vuole creare l'appello deve essere 1. Nel caso in cui questa condizione sia soddisfatta e vi sia svolto almeno un appello relativo ad una prova allora non saranno possibili ulteriori modifiche (insert, update o delete) relative al set di prove associate all'esame. E' dunque consentita una ripianificazione e conseguente modifica delle prove solo se non vi sono associati appelli o se non si è ancora verificato alcun appello relativo ad una delle prove. È importante sottolineare dunque che, al momento della creazione di un appello, l'esame e le relative prove sono considerati attentamente pianificati dai docenti.

- **Delete:**
  - Impedimento della cancellazione di una prova appartenente ad un esame tale per cui almeno una delle prove che lo costituiscono ha un appello associato.
  - Non si consente l'eliminazione di prove che sono provaPrimaria di altre prove, vale a dire non viene consentita l'eliminazione di prove il cui superamento è richiesto da altre. Questo vincolo è inserito per non permettere a delle Prove o a degli Appelli di riferirsi a qualcosa di non più esistente. Se si volesse in qualche modo effettuare una cancellazione si deve prima eliminare le eventuali prove collegate (ovvero le provaSuccessiva), e sarà compito degli insegnanti la gestione dei vari cambiamenti sulle varie prove collegate.

```
CREATE TRIGGER prevent_removal_of_trial_that_has_trials_associated
BEFORE DELETE ON Prove
FOR EACH ROW
EXECUTE FUNCTION
prevent_removal_of_trial_that_has_trials_associated_function();

CREATE OR REPLACE FUNCTION
prevent_removal_of_trial_that_has_trials_associated_function()
RETURNS TRIGGER AS $$
BEGIN
    IF( 1 <= SELECT COUNT (provaSuccessiva)
             FROM Superamenti
             WHERE provaPrimaria = OLD.codProva; ) THEN
    RETURN NULL;
    END IF;
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;
```

Trigger 2: controlla che non vi siano prove che richiedano come prova propedeutica una prova che si sta tentando di cancellare

```
CREATE TRIGGER prevent_removal_of_trial_that_has_round
BEFORE DELETE ON Prove
FOR EACH ROW
EXECUTE FUNCTION prevent_removal_of_trial_that_has_round_function();

CREATE OR REPLACE FUNCTION prevent_removal_of_trial_that_has_round_function()
RETURNS TRIGGER AS $$
BEGIN
    IF( 1 <= SELECT COUNT(*)
             FROM Appelli JOIN Prove ON prova = codProva
             WHERE docente = OLD.docente AND prova = OLD.codProva
                  AND data < CURRENT_DATE; ) THEN
    RETURN NULL;
    END IF;
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;
```

Trigger 3: controlla che non vi sia un appello relativo ad una prova che si sta cercando di cancellare

- Eventuali cancellazioni (consentite se non infrangono i vincoli precedentemente indicati) di una prova sono consentite ad un docente solo su quelle da lui stesso create.

- **Update:** Consentite modifiche fino a quando non ci sono appelli relativi alla prova che si vuole modificare
- **Insert:** E' fondamentale che due test dipendenti mediante la relazione superamenti facciano parte dello stesso esame, ovvero se ho una provaPrimaria e una provaSuccessiva allora queste devono essere relative al medesimo esame.

```
CREATE TRIGGER check_same_exam
BEFORE INSERT ON Superamenti
FOR EACH ROW
EXECUTE FUNCTION check_same_exam_function();

CREATE OR REPLACE FUNCTION check_same_exam_function()
RETURNS TRIGGER AS $$
BEGIN
    IF((SELECT esame
        FROM Superamenti JOIN Prove on codProva = provaSuccessiva )
        NOT IN
       (SELECT esame
        FROM Superamenti JOIN Prove on codProva = provaPrimaria )) THEN
    RETURN NULL;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Trigger 4: controlla che una prova e le sue relative prove propedeutiche siano relative al medesimo esame

ISCRIZIONI

- **Delete:** Agli studenti è consentita la dis-iscrizione da tutti gli appelli che non si sono ancora svolti. Quindi viene impedita la delete su Iscrizioni nel caso in cui l'iscrizione sia relativa ad un appello che si è già verificato.
- **Update:**
  - L'inserimento della valutazione da parte di un docente relativa ad uno studente che ha sostenuto un appello viene consentito solamente dopo che l'appello si è verificato (data > currentDate)

```
CREATE TRIGGER prevent_addition_of_score
BEFORE UPDATE ON Iscrizioni
FOR EACH ROW EXECUTE FUNCTION prevent_addition_of_score_function();

CREATE OR REPLACE FUNCTION prevent_addition_of_score_function()
RETURNS TRIGGER AS $$
BEGIN
    IF(CURRENT_DATE < SELECT data
                      FROM Appelli
                      WHERE codAppello = NEW.appello; ) THEN
    RETURN NULL;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Trigger 5: impedisce ai docenti di inserire valutazioni agli studenti iscritti ad un appello prima che questo si sia verificato

- Bisogna rispettare il seguente vincolo: "Quando si supera una prova, il suo superamento ha una data che registra quando è stata sostenuta e una data che registra la sua scadenza. *Dopo la scadenza una prova non è più valida*: rimane nel sistema, ma dal punto di vista della registrazione é come se non fosse stata superata", con superamento intendiamo che l'iscrizione relativa ad un appello abbia isValid = TRUE ovvero che sia il tentativo più recente e che Voto >= 18 e che currentDate < dataDiScadenza; quest'ultima condizione viene controllata periodicamente (ogni giorno) mediante una funzione che eseguirà il controllo e un job per eseguire questa funzione al fine di controllare la data di scadenza. Questo assicurerà che

le prove con date di scadenza non conformi vengano automaticamente invalidate senza la necessità del verificarsi di eventi sulla tabella iscrizioni.

```
CREATE OR REPLACE FUNCTION check_expiration_date RETURNS VOID AS $$
BEGIN
    UPDATE Iscrizioni
    SET isValid = FALSE;
    WHERE dataScadenza < CURRENT_DATE;
END;
$$ LANGUAGE plpgsql;
```

- Inoltre sempre citando la consegna, bisogna gestire altre situazioni di invalidazione delle prove. "*Un'altra ragione di invalidazione di una prova è il sostenimento della stessa prova ad un appello successivo.* Se l'appello più recente è superato, allora il voto precedente si invalida e quello nuovo ne prende il posto. Se invece l'appello più recente non è superato, la prova precedente viene invalidata e si registra la nuova come un fallimento."

```
CREATE TRIGGER invalidate_trial
BEFORE INSERT ON Iscrizioni
FOR EACH ROW EXECUTE FUNCTION invalidate_trial_function();

CREATE OR REPLACE FUNCTION invalidate_trial_function() RETURNS TRIGGER AS $$
BEGIN
    UPDATE Iscrizioni
    SET isValid = FALSE
    WHERE studente = NEW.studente AND
          appello IN (SELECT codAppello
                      FROM Appelli JOIN Iscrizioni ON appello = codAppello
                      WHERE isValid = TRUE AND Voto >= 18 AND
                      prova IN (SELECT prova
                                FROM Appelli
                                WHERE CodAppello = NEW.appello;););
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Trigger 6: automatizzazione invalida di una prova già superata perché uno studente si è iscritto ad un appello relativo alla suddetta prova

- Il vincolo precedente crea delle problematiche in quanto lo studente iscrivendosi all'appello relativo ad una prova già superata invalida il tentativo precedente ma

1. allo stesso tempo deve anche invalidare l'esame a cui la prova apparteneva se tale esame era in condizione di essere formalizzabile. Quindi è necessario far sì che in caso di esame formalizzabile la modifica all'attributo isValid su Iscrizioni si ripercuota anche sull'attributo passato su FormalizzazioneEsami.
2. è necessario invalidare inoltre tutte le prove superate che richiedevano come superamento la prova che si vuole ritentare (ciò provoca ricorsione di chiamate del trigger).

Viene di seguito fornito il trigger che gestisce entrambe queste situazioni.

```
CREATE TRIGGER invalidate_exam_cascade
BEFORE UPDATE ON Iscrizioni
WHEN NEW.isValid = FALSE
FOR EACH ROW EXECUTE FUNCTION invalidate_exam_cascade_function();

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

-- rimozione esame che era formalizzabile, in quanto una prova che lo
    componeva non e' piu' valida

UPDATE FormalizzazioneEsami
SET passato = FALSE
WHERE studente = NEW.studente AND passato = TRUE
      AND codEsame IN (SELECT esame
                       FROM Appelli JOIN Prove ON codProva = prova
                       WHERE CodAppello = NEW.appello;);

-- gestione invalidamento prove che richiedevano il superamento di una prova
    che non e' piu' valida (produce ricorsione di chiamate)

UPDATE Iscrizioni
SET isValid = FALSE
WHERE studende = NEW.studente AND isValid = TRUE AND
      appello IN ( SELECT codAppello
                   FROM Appelli
                   WHERE prova IN ( SELECT provaSuccessiva
                                    FROM Superamenti JOIN Appelli
                                    ON prova = provaPrimaria
                                    WHERE codAppello = NEW.appello;););
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Trigger 7: invalidamento delle prove che richiedevano la prova (per cui ci si sta re-iscrivendo ad un appello) come propedeuticità e rimozione esame (a cui la prova appartiene) dagli esami formalizzabili in caso tale esame fosse formalizzabile

- **Insert:** Uno studente non può iscriversi a un appello correlato a una prova che richiede il superamento di prove non ancora superate dallo studente, in accordo con lo spunto della consegna: "è possibile definire dei vincoli sulle prove d'esame, richiedendo, ad esempio, che la discussione del progetto debba essere effettuata solo dopo che l'esame scritto è stato superato."

```
CREATE TRIGGER check_if_other_tests_are_required
BEFORE INSERT ON Iscrizioni
FOR EACH ROW EXECUTE FUNCTION check_if_other_tests_are_required();

CREATE OR REPLACE FUNCTION check_if_other_tests_are_required()
RETURNS TRIGGER AS $$
BEGIN
    IF( (SELECT COUNT(provaPrimaria)
         FROM Superamento
         WHERE provaSuccessiva IN (SELECT prova
                                   FROM Appelli
                                   WHERE codAppello = NEW.appello;);)
        <>
        (SELECT COUNT(prova)
         FROM Appelli JOIN Iscrizioni ON codAppello = appello
         WHERE studente = NEW.studente AND AND Voto >= 18
               AND isValid = TRUE AND prova IN
                                     (SELECT provaPrimaria
                                      FROM Superamento
                                      WHERE provaSuccessiva IN
                                     (SELECT prova
                                       FROM Appelli
                                       WHERE codAppello = NEW.appello;);))
        THEN
        RETURN NULL;
    END IF;
```

```
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Trigger 8: si impedisce agli studenti di iscriversi ad appelli relativi a prove che richiedono il superamento di prove non ancora superate

- Agli studenti è vietato iscriversi ad appelli quando l'esame correlato alle prove di tali appelli è già stato formalizzato.

```
CREATE TRIGGER dont_allow_new_trial
BEFORE INSERT ON Iscrizioni
FOR EACH ROW EXECUTE FUNCTION dont_allow_new_trial_function();

CREATE OR REPLACE FUNCTION dont_allow_new_trial_function() RETURNS TRIGGER AS
    $$
BEGIN
    IF( SELECT prova
        FROM Appelli
        WHERE NEW.appello = CodAppello;
        IN
        SELECT CodProva
        FROM Prove NATURAL JOIN FormalizzazioneEsami
        WHERE NEW.studente = studente ) THEN
        RETURN NULL;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Trigger 9: impedimento iscrizione ad appelli per cui l'esame relativo alla prova da svolgere è già stato formalizzato

- Agli studenti ovviamente è vietato inoltre di iscriversi ad appelli che si sono già svolti.

FORMALIZZAZIONE ESAMI

- **Insert / Delete:** Non sono consentiti inserimenti e cancellazioni su questa tabella dal momento che vengono gestite in automatico dal sistema e nessuno dei ruoli Studente o Docente è autorizzato a svolgere tali azioni.
- **Update:** - Si necessita di automatizzare il processo di caricamento di un voto relativo a un esame per uno studente, quando quest'ultimo ha superato tutte le prove ad esso associate. Il calcolo del voto avviene attraverso la media pesata dei voti delle prove, a cui possono essere aggiunti eventuali bonus. Oltre alla computazione del voto, è essenziale segnalare l'esame come superato. Queste modifiche devono essere apportate ogni volta che si verifica una modifica di voto su Iscrizioni, ad esempio, al caricamento dei voti per un appello da parte di un docente. Questo è particolarmente rilevante poiché uno studente potrebbe essere nella situazione di formalizzare l'esame dopo aver superato una prova (in quanto potrebbe avere superato ed avere valide tutte le prove relative all'esame). L'approccio proposto prevede che, dopo che un docente ha caricato i voti, vengano eseguiti i seguenti controlli per tutte le prove associate allo stesso esame della prova per cui sono stati caricati i voti: se tutte le prove sono superate (isValid = True, Voto $\geq$ 18 e dataScadenza $>$ currentDate), allora si procede con il calcolo del voto finale e si indica la prova come superata.

```
CREATE TRIGGER set_final_score
AFTER UPDATE ON Iscrizioni
FOR EACH ROW EXECUTE FUNCTION set_final_score_function();

CREATE OR REPLACE FUNCTION set_final_score_function() RETURNS TRIGGER AS $$
BEGIN
    -- si controlla che il numero di prove superate e valide dello studente
    coincida con il numero di prove obbligatorie che compongono l'esame, le
    prove bonus non vengono conteggiate non essendo obbligatorie
    IF (SELECT COUNT(cod)
        FROM Prove
        WHERE esame IN
                    ( SELECT esame
```

```
                        FROM Appelli
                        JOIN Prove ON prova = cod
                        WHERE codAppello = NEW.appello; )
                AND (idoneita = TRUE OR peso != 0.0);
        =
        SELECT COUNT(cod)
        FROM Appelli
        JOIN Iscrizioni ON codAppello = appello
        JOIN Prove ON prova = cod
        WHERE studente = NEW.studente
              AND voto >= 18 AND isValid = TRUE
              AND esame IN ( SELECT esame
                             FROM Appelli JOIN Prove ON prova = cod
                             WHERE codAppello = NEW.appello; )
              AND (idoneita = TRUE OR peso != 0.0);
        ) THEN

        UPDATE FormalizzazioneEsami
        SET voto = ( SELECT SUM(
                            (CASE WHEN voto = NULL THEN 0 ELSE voto END) *
                            peso +
                            (CASE WHEN bonus = NULL THEN 0 ELSE bonus END))
                     FROM Iscrizioni JOIN Appelli
                     ON appello = codAppello
                     WHERE studente = NEW.studente AND voto >= 18
                           AND isValid = TRUE
                           AND prova IN ( SELECT cod
                                          FROM Prove
                                          WHERE esame IN
                                          ( SELECT esame
                                            FROM Appelli
                                            JOIN Prove ON prova = cod
                                            WHERE
                                            NEW.appello = codAppello )
                                        )
                    ), passato = TRUE
        WHERE studente = NEW.studente
              AND esame IN ( SELECT esame
                             FROM Appelli JOIN Prove ON prova = cod
                             WHERE NEW.appello = codAppello
                           );
    END IF;
    RETURN NULL;
END;
$$ LANGUAGE plpgsql;
```

Trigger 10: automatizzazione dell'inserimento del voto nella tabella formalizzazioneEsami dopo che tutte le prove relative ad un esame sono state superate e modifica dell'attributo passato a TRUE segnalando quindi la possibilità di formalizzazione dell'esame

- Viene vietato ai docenti di intervenire manualmente nella modifica del voto finale.
- E' consentita solo la modifica dell'attributo formalizzato da parte di uno studente e soltanto se passato è TRUE, in tutti gli altri casi le modifiche manuali sono vietate.

2.7 Indici

Per determinare la convenienza dell'utilizzo di indici, abbiamo raccolto e analizzato i dati dopo il completamento della web app mediante pg_stat_statements. E' da considerare che l'impiego degli indici può comportare anche svantaggi, come un aumento dell'utilizzo dello spazio su disco o come il potenziale rallentamento delle operazioni di inserimento, aggiornamento e cancellazione data la necessità di mantenere gli indici coerenti con i dati nella tabella. Per ottenere informazioni in modo da effettuare le scelte più adatte al nostro caso abbiamo utilizzato la seguente query:

```
SELECT query,
       calls,
       (total_exec_time / 1000 / 60) as total_min,
       mean_exec_time as avg_ms,
       CASE
           WHEN query ILIKE '%SELECT%' THEN 'SELECT'
           WHEN query ILIKE '%INSERT%' THEN 'INSERT'
           WHEN query ILIKE '%UPDATE%' THEN 'UPDATE'
           WHEN query ILIKE '%DELETE%' THEN 'DELETE'
           ELSE 'UNKNOWN'
           END      AS operation_type
FROM pg_stat_statements
ORDER BY total_min DESC
```

Vengono restituite diverse informazioni chiave, tra cui:

- La query completa eseguita all'interno del database, fornendo una visione dettagliata delle operazioni eseguite nel contesto del sistema.
- Il numero totale di chiamate, che rappresenta quante volte la query è stata eseguita nel periodo di monitoraggio.
- Il tempo totale di esecuzione, espresso in minuti, offre una panoramica del carico cumulativo di tempo dedicato all'esecuzione della query.
- Il tempo medio di esecuzione, misurato in millisecondi, fornisce una stima del tempo medio richiesto per ogni singola esecuzione della query.
- Il tipo di operazione, suddividendo le query in categorie come "SELECT", "INSERT", "UPDATE", "DELETE", o assegnando l'etichetta "UNKNOWN" per le query che non rientrano in nessuna delle categorie specificate.

Questi dati sono fondamentali per comprendere le prestazioni delle query nel sistema, identificare eventuali punti critici e valutare l'efficacia degli indici da implementare. L'analisi di questi risultati consente di ottimizzare il sistema, migliorando le prestazioni complessive e affrontando eventuali problemi di efficienza nella gestione delle query.
Vengono presentati dei grafici che forniscono una visione dettagliata del numero di query eseguite durante i test, considerando la navigazione del sito, le operazioni di inserimento, modifica ed eliminazione dei dati di base. Queste operazioni includono la creazione e la modifica di esami, creazione di prove di diversa tipologia, l'aggiunta di professori e studenti, l'inserimento dei voti, la gestione degli appelli, l'eliminazione di appelli, l'invalidamento delle prove, la formalizzazione o il rifiuto dei voti finali associati agli esami. Si tenga in considerazione che le query sono state eseguite su:

- Numero di studenti: 5
- Numero di professori: 14
- Numero di esami: 11

Il totale delle query eseguite è stato di 3162, offrendo un quadro completo dell'attività del sistema durante i test.

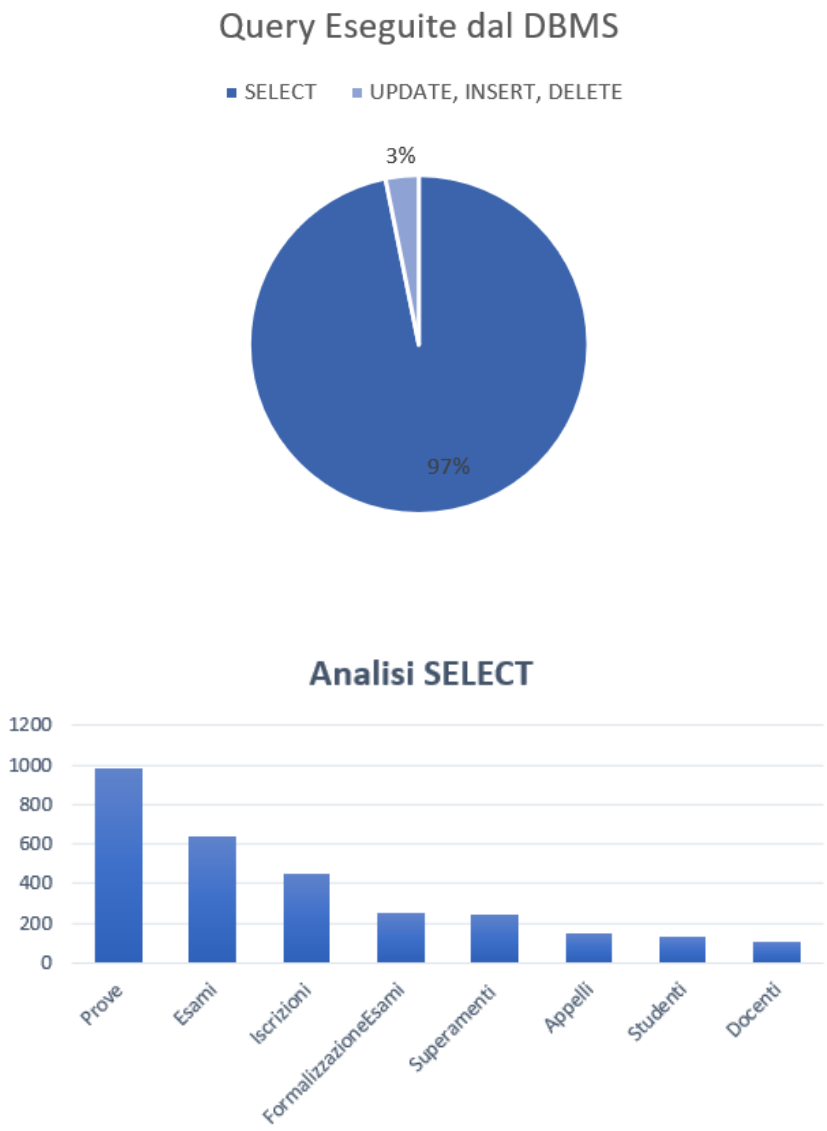

Figura 3: Rappresenta solo le SELECT con frequenza maggiore, non tutte quelle eseguite.

Dai grafici presentati emerge chiaramente che la maggior parte delle operazioni eseguite dal Database sono di tipo SELECT. Un'analisi attenta delle query ottenute (disponibili nel file Results.csv) rivela che la maggior quantità di queste query è basata sulle chiavi primarie delle diverse entità. Poiché Postgres, come indicato nella documentazione ufficiale, genera automaticamente indici sulle chiavi primarie, e considerando che, escludendo anche le query in cui i dati vengono combinati (con JOIN) utilizzando le chiavi primarie, il numero di query residue è esiguo, abbiamo deciso di non implementare ulteriori indici manualmente.
Questo approccio è giustificato dal fatto che il numero limitato di query residue, specialmente considerando che il sistema in fase di test non simula un numero realistico di docenti, esami e studenti, rende superflua la creazione di indici specifici per queste operazioni. In un contesto realistico con un numero significativo di entità, le operazioni di aggiornamento, inserimento e modifica risulterebbero più frequenti rispetto a quanto identificato in fase di test, seppur sempre minori rispetto alle query di selezione. L'introduzione di eventuali indici ottimizzati per selezioni meno rilevanti potrebbe causare rallentamenti nell'utilizzo realistico dell'applicazione. Da ciò possiamo concludere che gli indici automatici generati da Postgres sono sufficienti per massimizzare le prestazioni, garantendo un'efficienza ottimale nel filtraggio e nell'unione dei dati, senza la necessità di introdurre ulteriori indici specifici.

3 Sviluppo back-end

3.1 Scelta di Flask come framework

Nella fase iniziale del nostro progetto, ci siamo confrontati con la decisione cruciale di selezionare un framework per lo sviluppo dell'applicazione web. Dopo un'attenta valutazione delle opzioni disponibili e delle nostre competenze pregresse, abbiamo deciso di adottare Flask come il framework principale per la realizzazione del nostro progetto.

3.1.1 Familiarità e competenza

La nostra decisione di optare per Flask è stata influenzata dalla nostra familiarità con questo framework. Dato che questa costituiva la nostra prima esperienza nello sviluppo di applicazioni web, abbiamo ritenuto prudente utilizzare un framework con cui eravamo già familiari. La scelta di Flask è stata motivata anche dalla formazione approfondita fornita dal professore durante le lezioni, fornendoci una solida base di conoscenze su cui costruire il nostro progetto.

3.1.2 Adattabilità e flessibilità

Flask si è rivelato una scelta ideale per il nostro progetto grazie alla sua natura leggera e flessibile. La struttura minimale di Flask ci ha permesso di concentrarci direttamente sullo sviluppo delle funzionalità chiave della nostra applicazione senza dover gestire complessi aspetti del framework che avrebbero potuto appesantire il processo di apprendimento.

3.1.3 Funzionalità offerte da Flask

Dopo un'approfondita analisi delle caratteristiche di Flask, siamo rimasti impressionati dalle sue funzionalità robuste e dalla sua capacità di gestire efficacemente le sfide dello sviluppo web. Flask offre una vasta gamma di estensioni e moduli che hanno facilitato l'implementazione di funzionalità avanzate e la gestione efficiente delle richieste HTTP.
In conclusione, la scelta di Flask come framework per il nostro progetto è stata il risultato di una ponderata riflessione sulle nostre competenze, sulla formazione ricevuta in classe e sulla versatilità offerta da questo framework. Siamo convinti che questa scelta ci abbia consentito di sviluppare un'applicazione web robusta e performante, rispondendo al meglio alle esigenze del nostro progetto.

3.2 Struttura dell'applicazione

Il sistema presenta una gerarchia di directory ben strutturata, agevolando una suddivisione coerente dei file e delle risorse. Qui di seguito, forniamo un dettaglio dell'albero delle directory principali:

3.2.1 Struttura della directory e dei moduli

```
|-- /ExamManagementApp
|   |-- /App
|   |   |-- /blueprint
|   |   |   |-- auth.py
|   |   |   |-- admin.py
|   |   |   |-- student.py
|   |   |   |-- teacher.py
|   |   |-- /db
|   |   |   |-- /models
|   |   |   |   |-- database.py
|   |   |   |-- create_db.py
|   |   |-- /static
|   |   |   |-- sidebar.css
|   |   |   |-- ...
|   |   |-- /templates
|   |   |   |-- /admin
|   |   |   |   |-- homeAdmin.html
|   |   |   |   |-- ...
|   |   |   |-- /student
|   |   |   |   |-- homeStudente.html
|   |   |   |   |-- ...
|   |   |   |-- /teacher
|   |   |   |   |-- homeTeacher.html
|   |   |   |   |-- ...
|   |   |-- error.html
|   |   |-- home.html
|   |   |-- login.html
|   |   |-- /utils
|   |   |   |-- utilies.py
|   |   |-- __init__.py
|   |   |-- checkFunctions.py
|   |-- /Doc
|   |   |-- report.tex
|   |-- /out
|   |   |-- ..
|   |-- /venv
|   |   |-- ..
|   |-- .env
|   |-- main.py
```

3.2.2 Componenti chiave

La struttura dei file è stata accuratamente progettata per creare un ambiente di sviluppo ordinato e pulito, fondamentale per una gestione efficiente del progetto.
Il fulcro della directory è rappresentato dalla cartella "App", che contiene l'intera logica dell'applicazione. All'interno di questa cartella, troviamo moduli chiave che giocano un ruolo fondamentale nella struttura e funzionalità complessiva dell'applicazione.

- **blueprint:** Questa directory si specializza nella gestione della modularità dell'applicazione. I file blueprints, come *auth.py*, *student.py*, *admin.py*, e *teacher.py*, sono progettati per riflettere una suddivisione logica delle funzionalità. Questo approccio modulare semplifica significativamente la gestione e l'estensione delle funzionalità dell'applicazione.
- **db:** Questa directory è dedicata alla definizione dei modelli del database e alla loro creazione. Al suo interno, troviamo il file *database.py* per la definizione del modello del database e *createdb.py* per la creazione effettiva del database. Tale struttura agevola notevolmente la gestione delle operazioni sul database.

- **static e templates:** Le directory static e templates contengono risorse statiche ,come immagini o stylesheets css, e modelli HTML rispettivamente, garantendo una distinta separazione tra il contenuto dinamico e statico dell'applicazione. Le directory sono organizzate in maniera logica, suddividendo i file nelle cartelle "admin", "studente" e "docente". Questa struttura facilita la gestione ed estensione dell'applicazione, definendo chiaramente le responsabilità di ciascuna sezione in base ai ruoli degli utenti.

- **utils:** Questa directory ospita *utilities.py,* un modulo appositamente progettato per incorporare funzioni di utilità comuni. La sua finalità principale è di migliorare la riutilizzabilità del codice, semplificando la gestione di operazioni comuni all'interno dell'applicazione. Inoltre, contribuisce a ottimizzare il processo di individuazione e risoluzione degli errori, fornendo un punto centralizzato per le funzioni di utilità.

  La struttura dei file è stata progettata per creare un ambiente di sviluppo pulito e ordinato, fondamentale per una gestione efficiente del progetto. Alcuni punti salienti includono:

- Il modulo *init.py* inizializza l'applicazione prima dell'avvio del server.

- Il modulo *checkFunctions.py* contiene la definizione di decoratori utili per garantire la sicurezza del database.

- Infine, *main.py* rappresenta il punto cruciale di avvio dell'applicazione, attivando il server Flask e rendendo l'applicazione pronta per l'elaborazione delle richieste in arrivo.

  In sintesi, questa struttura ben organizzata facilita la manutenzione e l'espansione del progetto, fornendo una solida base per lo sviluppo e la collaborazione efficace tra i membri del team.

3.3 Sicurezza

Abbiamo posto grande attenzione all'implementazione di robusti meccanismi di sicurezza per garantire la protezione dei dati e delle operazioni all'interno dell'applicazione.

3.3.1 Autenticazione e Autorizzazione

Per garantire un accesso controllato alle risorse, abbiamo implementato un sistema di autenticazione solido e affidabile. Gli utenti devono autenticarsi utilizzando credenziali valide attraverso il nostro sistema di autenticazione, che sfrutta la potenza e la flessibilità di Flask-Login. Questo componente aggiuntivo semplifica la gestione delle sessioni utente e fornisce un meccanismo sicuro per l'autenticazione.

Flask-Login
Abbiamo integrato Flask-Login nel nostro sistema per gestire le sessioni utente in modo efficiente. Questa estensione semplifica la gestione dello stato di autenticazione, consentendo agli utenti di accedere alle funzionalità dell'applicazione solo dopo aver fornito credenziali valide.

Esempio di Codice - Flask-Login
Nel seguente frammento di codice, presentiamo un esempio illustrativo su come: inizializzare Flask-Login, definire una classe utente, implementare la funzione di caricamento dell'utente e utilizzare loginUser per gestire l'accesso. È fondamentale sottolineare che questo esempio è semplificato e non riflette completamente la complessità della logica di login all'interno della nostra applicazione. La decisione di strutturare il codice in moduli separati è stata presa per ottimizzare la gestione complessiva del progetto, richiedendo quindi alcune personalizzazioni specifiche alle esigenze particolari del nostro contesto.

```
1 from flask_login import LoginManager, UserMixin, login_user, login_required
      , logout_user, current_user
2
3 # Inizializzazione di Flask-Login
4 login_manager = LoginManager(app)
5 login_manager.login_view = 'login'
6
```

```python
# Definizione della classe Utente
class User(UserMixin):
    pass

# Funzione per caricare un utente
@login_manager.user_loader
def load_user(user_id):
    # Implementazione per caricare l'utente da un sistema di autenticazione
    return User.get(user_id)

# Esempio di utilizzo di login_user per effettuare l'accesso
@app.route('/login', methods=['POST'])
def login():
    # Logica per verificare le credenziali e ottenere l'utente autenticato
    user = get_authenticated_user()
    login_user(user)
    return redirect(url_for('dashboard'))
```

Ruoli e Autorizzazione
Per affinare ulteriormente il controllo degli accessi, abbiamo implementato un sistema di autorizzazione basato su ruoli. Ad ogni utente è assegnato un ruolo, determinando così le operazioni consentite. Ad esempio, il ruolo "Docente" ha privilegi diversi rispetto al ruolo "Studente". In questo modo, le azioni eseguibili sul database e le risorse sono strettamente limitate e controllate, garantendo un ambiente sicuro e conforme alle specifiche esigenze di autorizzazione.

Esempio di Codice - assegnazione di un Ruolo
Nel seguente esempio di codice, viene illustrata la logica di assegnazione di un ruolo dopo il login dell'utente. È importante notare che questo esempio è semplificato e non riflette l'implementazione reale, ma solo la logica di base.

```python
# loggo l'utente, di default la connessione e' per utente
login_user(pair[1])
# taglio la connessione
db.session.close()
if session['user_type'] == Studenti.__name__:
# ristabilisco la connessione
    current_app.config['SQLALCHEMY_DATABASE_URI'] = os.getenv('
    DATABASE_URL_LOCALE_STUDENTE')
    return redirect(url_for('student.studentPage'))
else:
# ristabilisco la connesione
    current_app.config['SQLALCHEMY_DATABASE_URI'] = os.getenv('
    DATABASE_URL_LOCALE_DOCENTE')
    return redirect(url_for('teacher.teacherPage'))
```

3.3.2 Decorator

I decorator in Python sono funzioni speciali che consentono di estendere o modificare il comportamento di un'altra funzione o metodo. Nell'applicazione, i decoratori sono stati sfruttati per garantire un accesso controllato ai vari endpoint.

Flask-Login, Decorator
Il decorator @login-required fornito da Flask-Login è utilizzato per proteggere le route o le funzioni nelle applicazioni Flask, richiedendo che l'utente sia autenticato prima di poter accedere a quelle risorse. Questo decorator svolge un ruolo chiave nel garantire che solo gli utenti autenticati possano accedere a determinate parti dell'applicazione.

Decoratori personalizzati
E' emersa la necessità di implementare due decoratori dedicati al controllo degli accessi. Le funzioni

checkStudente, checkDocente e checkAdmin sono stati creati come decoratori in Python, progettati per proteggere specifiche route o funzioni all'interno di un'applicazione. Forniscono inoltre un accesso controllato alle risorse dell'applicazione, verificando le credenziali dell'utente in modo personalizzato. Mentre checkStudente si occupa di verificare se l'utente è di tipo Studente, checkDocente controlla se l'utente è un Docente e checkAdmin se l'utente è l'admin. Utilizzando questi decoratori nelle route o funzioni pertinenti, è possibile applicare un controllo granulare sull'accesso, adattandolo alle specifiche esigenze del progetto.
L'implementazione di tali decoratori non solo consente una gestione più efficace dell'accesso controllato, ma contribuisce anche a evitare la ripetizione di istruzioni condizionali come "if session['usertype'] == ?" in ogni endpoint dell'applicazione. Questo approccio non solo riduce la complessità del codice, ma anche il rischio di errori dovuti a omissioni o dimenticanze nel controllo dell'accesso, migliorando così la robustezza e la manutenibilità dell'applicazione.

```
def checkDocente(function):
    @wraps(function)
    def wrapper(*args, **kwargs):
        if not isinstance(current_user, Docenti):
            return render_template('error.html', error='Non hai accesso a questa pagina')
        return function(*args, **kwargs)

return wrapper


def checkStudente(function):
    @wraps(function)
    def wrapper(*args, **kwargs):
        if not isinstance(current_user, Studenti):
            return render_template('error.html', error='Non hai accesso a questa pagina')
        return function(*args, **kwargs)

    return wrapper


def checkAdmin(function):
    @wraps(function)
    def wrapper(*args, **kwargs):
        if not isinstance(current_user, Admin):
            return render_template('error.html', error='Non hai accesso a questa pagina')
        return function(*args, **kwargs)

    return wrapper
```

Esempio Applicativo
Nel seguente esempio, il decorator @login-required garantisce che l'utente sia autenticato, mentre i decoratori personalizzati @checkStudente e @checkDocente verificano l'appartenenza dell'utente alle rispettive categorie.

```
@student.route('/')
@login_required
@checkStudente
def studentPage():
    return render_template('student/home.html', student=current_user)
```

Questo approccio contribuisce a garantire la sicurezza e la conformità alle regole specifiche dell'utente.

3.3.3 Protezione da SQL Injection e XSS

La sicurezza delle applicazioni web è una priorità fondamentale, e framework come Flask-SQLAlchemy integrano funzionalità di sicurezza per mitigare potenziali minacce, come l'iniezione di SQL.

Protezione da SQL Injection con Flask-SQLAlchemy e Parameterized Queries di SQLAlchemy
L'iniezione di SQL rappresenta una minaccia potenziale in cui un attaccante inserisce intenzionalmente codice SQL dannoso all'interno delle query, sfruttando input non validato per compromettere la sicurezza del database. Flask-SQLAlchemy, in collaborazione con il meccanismo di Parameterized Queries di SQLAlchemy, affronta con decisione questo rischio.
Il concetto fondamentale di Parameterized Queries consiste nell'evitare la concatenazione diretta di stringhe nella query SQL. Invece di incorporare i valori direttamente nella stringa SQL, i parametri vengono utilizzati per inserire dinamicamente i valori nella query. Questo approccio introduce un livello aggiuntivo di sicurezza, impedendo l'esecuzione di codice SQL non desiderato.
Flask-SQLAlchemy, sfruttando questo meccanismo, consente di costruire query SQL in modo sicuro, garantendo che i parametri vengano trattati correttamente e prevenendo efficacemente le iniezioni di SQL. La combinazione di Flask-SQLAlchemy e Parameterized Queries di SQLAlchemy rappresenta quindi un robusto sistema di difesa contro le potenziali vulnerabilità legate alle iniezioni di SQL, contribuendo a garantire la sicurezza delle operazioni di database nell'ambito dell'applicazione.

Esempio di Query parametrizzata

```
# Esempio di query parametrizzata
username_input = "Domenico"
query = Studente.query.filter_by(username=username_input).first()
```

Protezione da XSS con Jinja (Template Engine di Flask)
Cross-Site Scripting (XSS) è un'altra minaccia comune che può verificarsi quando l'applicazione consente l'inserimento non sanificato di codice JavaScript all'interno delle pagine visualizzate da altri utenti. Flask utilizza Jinja come motore di template, e Jinja offre misure di sicurezza per mitigare il rischio di XSS. Jinja implementa l'escape automatico dei dati di output, convertendo caratteri speciali in entità HTML, prevenendo così l'esecuzione non desiderata di script. Ad esempio, se un utente inserisce un testo contenente script JavaScript all'interno di un campo di input, Jinja si assicura che tali script vengano resi inoffensivi quando visualizzati da altri utenti. Questa pratica riduce il rischio di XSS, garantendo che il contenuto dinamico generato dalle pagine web sia trattato in modo sicuro e non rappresenti una minaccia per la sicurezza complessiva dell'applicazione.

3.4 Interazione con il DBMS

L'interazione con il DBMS costituisce un elemento fondamentale per garantire la persistenza, l'accessibilità, la coerenza e la sicurezza dei dati all'interno di un'applicazione.

3.4.1 Utilizzo di SQLAlchemy

Nel contesto dell'applicazione, l'interazione con il sistema di gestione del database (DBMS) PostgreSQL è gestita mediante l'utilizzo della libreria SQLAlchemy.

Perchè SQLAlchemy? La decisione di adottare SQLAlchemy come ORM (Object-Relational Mapping) nel nostro progetto è stata influenzata da molteplici fattori. In primo luogo, SQLAlchemy fornisce un'astrazione solida e flessibile per la gestione delle operazioni del database, agevolando una transizione fluida tra diversi motori di database e semplificando la creazione di query complesse. La sua struttura modulare e la capacità di gestire le relazioni tra oggetti contribuiscono significativamente a semplificare lo sviluppo e la manutenzione del codice.

Un ulteriore motivo della nostra scelta è la diffusa adozione di SQLAlchemy nella comunità Python, il che garantisce un supporto robusto e una vasta documentazione. Inoltre, la familiarità con questa tecnologia è stata rafforzata dalla spiegazione dettagliata fornita dal professore durante le lezioni.

3.4.2 Query principali e gestione mediante metodi

L'applicazione offre una variegata gamma di query, ciascuna con una funzione specifica nell'interazione con il database. La gestione di tali operazioni è organizzata attraverso una serie di metodi distribuiti tra diverse classi, assicurando un approccio organizzato e modulare nel codice.
Particolarmente rilevanti sono alcuni metodi chiave presenti all'interno delle classi `Prove`, `Docenti`, `Esami` e `Studenti`. Questi metodi sono progettati per svolgere ruoli cruciali all'interno delle rispettive entità e contribuiscono in modo significativo alla funzionalità complessiva dell'applicazione.

Classe Prove
Metodo che data una prova, ritorna le prove che la richiedevano come prova propedeutica.

```
def getProveSuccessive(self, codProva):
    query = db.session.query(Superamenti.provaSuccessiva).filter(
    Superamenti.provaPrimaria == codProva)
    return query.all()
```

Metodo che data una prova, ritorna le prove propedeutiche.

```
def getProvePrecedenti(self, codProva):
    query = db.session.query(Superamenti.provaPrimaria).filter(Superamenti.
    provaSuccessiva == codProva)
    return query.all()
```

Classe Docenti
Metodo che dato un numero di matricola, un voto e il codice di un appello, carica il voto allo studente sulla relativa iscrizione.

```
def set_voto_prova(self, studente_matricola, voto, codAppello):
    voto = int(voto)
    isValid = False
    if voto >= 18 and voto <= 31 and voto != None:
        isValid = True
    with db.engine.begin() as connection:
        # Execute the update statement
        connection.execute(
            iscrizioni.update()
            .where((iscrizioni.c.studente == studente_matricola) & (
    iscrizioni.c.appello == codAppello))
            .values({'voto': voto, 'isValid': isValid})
            )
    # Commit the changes to the database
    db.session.commit()
```

Classe Studenti
Metodo che dato il codice di un esame, ritorna il voto se esiste, none altrimenti.

```
def getVotoEsame(self, codEsame):
    # Build the SQL query
    query = (
        select(
            formalizzazioneEsami.c.voto
        )
        .where((formalizzazioneEsami.c.studente == self.matricola) &
                    (formalizzazioneEsami.c.voto != None) &
```

```
                    (formalizzazioneEsami.c.esame == codEsame))
     )
    with db.engine.connect() as connection:
        result = connection.execute(query)
    res = result.fetchall()
    if res[0][0]:
        return res[0][0]
    else:
        return none
```

Metodo che dato il codice di un esame, lo formalizza, settando "formalizzato" a true.

```
def formalizzaEsame(self, codEsame):
    with db.engine.begin() as connection:
        # Execute the update statement
        connection.execute(
            formalizzazioneEsami
            .update()
            .where(
                (formalizzazioneEsami.c.studente == self.matricola) & (
                            formalizzazioneEsami.c.esame == codEsame))
            .values({'formalizzato': True})
        )
```

Metodo che ritorna i voti finali relativi ad esami formalizzati da un dato studente

```
def getEsamiFormalizzati(self):
    # Build the SQL query
    query = (
        select(
            formalizzazioneEsami.c.esame
        )
        .where((formalizzazioneEsami.c.studente == self.matricola) & (
                    formalizzazioneEsami.c.formalizzato == True) &
                    (formalizzazioneEsami.c.voto != None))
    )

       # Execute the query
       with db.engine.connect() as connection:
        result = connection.execute(query)

    res = result.fetchall()

    esami = []
    for i in range(len(res)):
        esami.append(Esami.query.get(res[i][0]))

    return esami
```

Metodo che dato il codice in un esame, lo rifiuta, settando a false "formalizzato" e il voto a None.

```
def rifiutaEsame(self, codEsame):
    with db.engine.begin() as connection:
        # Execute the update statement
        connection.execute(
            formalizzazioneEsami
            .update()
            .where(
                (formalizzazioneEsami.c.studente == self.matricola) & (
                            formalizzazioneEsami.c.esame == codEsame))
            .values({'voto': None, 'formalizzato': False})
        )
```

```
        appello = Appelli.query.get(codAppello)
```

```
        studente = Studenti.query.filter_by(email=email).first()
        prova = Prove.query.filter_by(cod = 'BDProject').first()
```

Classe Esami

Metodo che recupera il voto assegnato a uno studente per l'esame corrente, "stabbando" una funzione già definita per gli studenti. L'uso di questa tecnica favorisce il riutilizzo del codice esistente e contribuisce a mitigare il rischio di errori.

```
def getVoto(self, matricola):
    studente = Studenti.query.get(matricola)
    return studente.getVotoEsame(self.cod)
```

4 Implementazione front-end

Il front-end rivolge la sua attenzione in modo specifico agli utenti finali, con l'obiettivo primario di rendere positiva la loro esperienza. Questa fase dello sviluppo è focalizzata quindi sulla creazione di interfacce piacevoli e semplici da usare. Inoltre, attribuiamo grande importanza alla comunicazione visuale degli eventi. Ad esempio, utilizziamo bottoni disabilitati e messaggi correlati per trasmettere chiaramente le azioni che si verificano.

4.1 Tecnologie utilizzate

Sono state utilizzate diverse tecnologie front-end di base e framework per lo sviluppo della nostra web application, in particolare:

- **HTML (HyperText Markup Language):** per definire la struttura di base delle pagine web e dei suoi componenti abbiamo usato questo linguaggio di markup standard.
- **CSS (Cascading Style Sheets):** usato per rendere coerenti tutte le pagine web tra di loro, personalizzando a nostra discrezione gli attributi dei componenti HTML come colori, dimensioni e font.
- **JavaScript**: Dove si è reso necessario sono state implementate delle funzioni di JavaScript, sia per rendere responsivi alcuni elementi, come il tasto menu, sia per poter comunicare all'utente risultati forniti dal database, come quando si mostra allo studente la data di svolgimento di uno specifico esame. In generale è stato utilizzato ogni qualvolta servisse un cambiamento in tempo reale.
- **Bootstrap:** Al fine di rendere visivamente gradevole l'app, si sono ampiamente usati i temi e componenti predefiniti forniti da Bootstrap. Abbiamo cercato di mantenere l'utilizzo di una sola versione di Bootstrap (in particolare la quarta versione), ma per sfruttare alcuni nuovi componenti introdotti nella quinta, abbiamo scelto per due pagine di cambiare versione. L'utilizzo mirato di Bootstrap 5 in queste aree ci consente di beneficiare delle nuove funzionalità senza dover riscrivere completamente il codice. Nel caso fossero sorte incongruenze, ad esempio di colore, tra due pagine che utilizzano due versioni differenti, abbiamo apportato manualmente modifiche, cioè abbiamo aggiunto in HTML gli attributi di stile CSS mancanti.

4.2 Scelte di design

Le pagine web sono organizzate in modo da essere tutte coerenti, quindi usano tutte gli stessi componenti, container, bottoni, layout. Per quanto riguarda i colori ci siamo affidati principalmente a quelli predefiniti di bootstrap 4, identificati come "info" e "danger", rispettivamente azzurro e rosso. Inoltre si è deciso di rendere molto agevolata la navigazione degli utenti sulla nostra web application. A questo scopo due sono le funzionalità principali presenti in ogni pagina del sito web, a meno delle pagine iniziali di home e login, dove l'utente deve essere prima reindirizzato nel proprio profilo perchè si possa decidere quali operazioni può compiere. Le due funzioni sono:

- Una barra di navigazione che mostra i titoli di tutte le pagine già navigate. Queste sono interagibili per poter tornare alla pagina precedente. Inoltre, aiuta a mantenere uno stile specifico per tutte le schermate.
- Un tasto menu, integrato nella barra di navigazione, che replica tutte le funzionalità di cui dispone l'utente in modo che possa spostarsi facilmente da una all'altra senza dover ritornare alla home.

4.3 Integrazione tra front-end e back-end

Il nostro front-end comunica con il back-end utilizzando richieste HTTP come GET e POST. Questo è un approccio molto comune per effettuare scambi di dati tra le due parti. In questo modo l'utente può inviare i suoi dati, come quando riempie form relativi a prove da creare, e interagire con il database, richiedendo ad esempio di vedere tutti gli studenti iscritti a una prova. Nel file HTML si specifica, per una certa azione, quale richiesta può gestire, se GET, POST o entrambe.

Successivamente a lato back-end le richieste HTTP sono gestite attraverso il modulo Flask, controllando le route definite con i decorator, e qual è il tipo di richiesta. Di seguito forniamo degli esempi di come sono stati usati queto tipo di richieste nella nostra applicazione.

Esempio di GET

```
@teacher.route('/visualizzaAppelli/dettagliAppello/<codAppello>', methods=['GET'])
@login_required
@checkDocente
def visualizzaDettagliAppello(codAppello):

        return render_template('teacher/visualizzaDettagliAppello.html', appello=Appelli.query.get(codAppello))
```

Dal file HTML visualizzaDettagliAppello:

```
<div class="card">
    <div class="card-body p-3">
        <h5 class="card-title">Codice Appello: {{ appello.codAppello }}</h5>
        <p class="card-text">Data Appello: {{ appello.data.strftime('%Y-%m-%d')
          }}</p>
        <p class="card-text">Ora Appello: {{ appello.data.strftime('%H:%M') }}</p>
        <p class="card-text">Esame: {{ appello.prove.esami.name }}</p>
        <p class="card-text">Prova: {{ appello.prove.cod }}</p>
        <p class="card-text">Docente: {{ appello.prove.docenti }}</p>
    </div>
</div>
```

Dove avvengono i seguenti passaggi:

- La route '/visualizzaAppelli/dettagliAppello/¡codAppello¿' è associata al metodo GET. Questo significa che la funzione visualizzaDettagliAppello verrà eseguita quando viene effettuata una richiesta GET a questa specifica route.
- Nella funzione visualizzaDettagliAppello, viene preso ¡codAppello¿ e passato come argomento alla funzione. Questo consente di utilizzare il valore di codAppello all'interno della funzione.
- La funzione restituisce una pagina HTML (visualizzaDettagliAppello.html) renderizzata utilizzando renderTemplate. All'interno del template, i dati dell'appello sono accessibili tramite la variabile appello, che è passata come argomento nel renderTemplate.
- Nel file HTML (visualizzaDettagliAppello.html) utilizziamo le variabili Jinja appello.proprietà per inserire dinamicamente i dati dell'appello all'interno della pagina web.

Esempio di POST

```
@teacher.route('/visualizzaAppelli/studentiIscritti/setVoto', methods=['POST'])
@login_required
@checkDocente
def setVoto():
    if request.method == 'POST':
        voto = request.form['voto']
        studente_id = request.form['studente_id']
        codAppello = request.form['codAppello']
        current_user.set_voto_prova(studente_id, voto, codAppello)

    return redirect(url_for('teacher.visualizzaStudentiIscritti', codAppello=codAppello))
```

Dal file HTML visualizzaStudentiIscritti:

```
<form method="post" action="{{ url_for('teacher.setVoto') }}" onsubmit="return
validateForm()">
    <input type="hidden" name="studente_id" value="{{ studente.matricola }}">
    <input type="hidden" name="codAppello" value="{{ codAppello }}">
    <div class="form-group">
        <input type="number" name="voto" id="voto" class="form-control"
          placeholder="Inserisci il voto" required min="0" max="31">
    </div>
    <button type="submit" class="btn btn-danger">Imposta Voto</button>
</form>
```

Dove avvengono i seguenti passaggi:

- La pagina HTML contiene un modulo che invia una richiesta post quando viene premuto il bottone ”Imposta voto”
- La route ’/visualizzaAppelli/studentiIscritti/setVoto’ viene associata al metodo POST dichiarando: methods=[’POST’]. Ciò significa che la funzione setVoto verrà eseguita solo quando viene effettuata una richiesta HTTP di tipo POST a questa route specifica. Ciò è rafforzato dal controllo eseguito nella funzione setVoto, con if request.method == ’POST’:. Ciò garantisce che la logica all’interno di questa funzione venga eseguita solo quando viene inviata una richiesta POST.
- I dati vengono recuperati dalla richiesta POST con request.form[’nomeCampo’], dove il nomecampo è il voto, studentID e codice appello, e con questi dati si invoca la funzione setVotoProva di currentUser, che è cioè il docente.
- Infine reindirizziamo l’utente alla pagina visualizzaStudentiIscritti utilizzando un redirect.

5 Contributo al progetto

5.1 Suddivisione compiti

Nella fase iniziale dello sviluppo della web app il gruppo si è concentrato sul creare una base solida di un database da cui partire. A tale scopo i componenti hanno lavorato assieme alla stesura di uno schema il più completo possibile scambiandosi versioni prima fatte in autonomia. Successivamente è stata effettuata una suddivisione più netta, ma non assoluta, cercando di rispecchiare i tre livelli di una web-app, cioè:

1. Presentation tier, anche detto front-end, compito di Eleonora Greco
2. Logic tier, anche detto back-end, compito di Domenico Sosta
3. Data tier, compito di Donald Gera

E' importante sottolineare che, nonostante la suddivisione delle attività sia stata definita per indicare le principali responsabilità di ciascun membro del team nei rispettivi tier, ogni componente ha contribuito in modo significativo a tutte le varie parti del progetto. La divisione delle attività è quindi solamente indicativa di chi ha contribuito in una percentuale maggiore a una determinata attività, ma non esclude il coinvolgimento attivo di tutti nei diversi aspetti del progetto.
Altri compiti sono invece stati pensati come lavori di gruppo e quindi svolti in presenza con l'obiettivo di interessare tutti, quali:

- Documentazione
- Log e Analisi delle Prestazioni degli Indici
- Video Dimostrativo

Questa metodologia ha promosso un ambiente di lavoro collaborativo, consentendo a tutti i membri del team di contribuire in modo significativo a tutte le fasi del progetto.

5.2 Piattaforme di collaborazione

Il gruppo si è impegnato a mantenere una costante comunicazione e collaborazione tramite incontri fisici per incorporare i codici scritti o piattaforme di videochiamate per problemi più urgenti che avrebbero bloccato lo sviluppo.
Per quanto riguarda la redazione della documentazione è stato usato un editor online di LaTeX, ovvero "Overleaf", a fine di ottenere un documento aggiornato in tempo reale e permettere una scrittura simultanea da parte dei componenti.
Per agevolare la stesura di parti differenti di codice si è deciso di usare GitHub, che offre la possibilità di creare branch differenti su cui eseguire successivamente l'operazione di merge e risoluzione conflitti ove necessario. Di seguito l'analitica GitHub che delinea la durata temporale dedicata all'esecuzione del progetto e i contributi collettivi alla scrittura del codice, quindi sia di front-end che back-end.

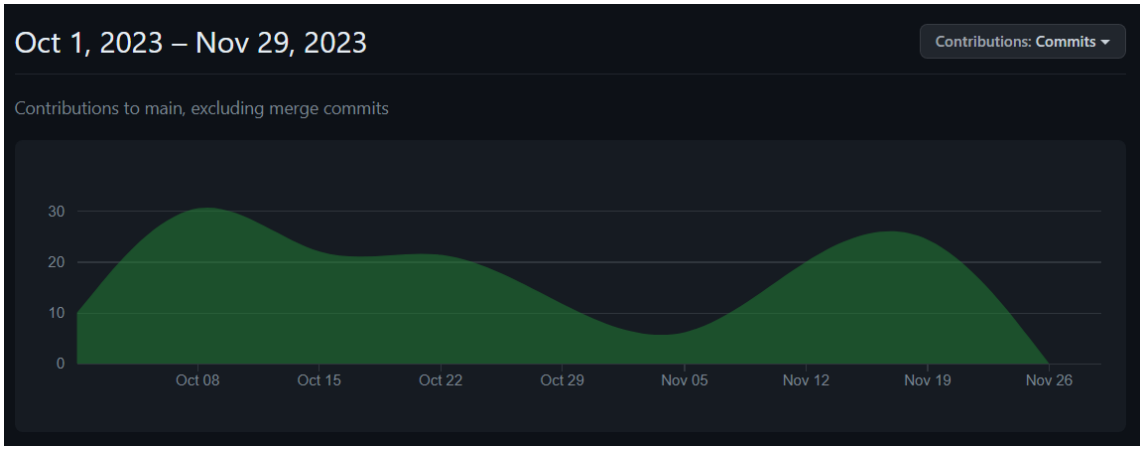

Figura 4: Commit GitHub