

# PiGEU

## Piattaforma per la Gestione degli Esami Universitari

### DOCUMENTAZIONE TECNICA

Fontana Francesco  
matr. 943519

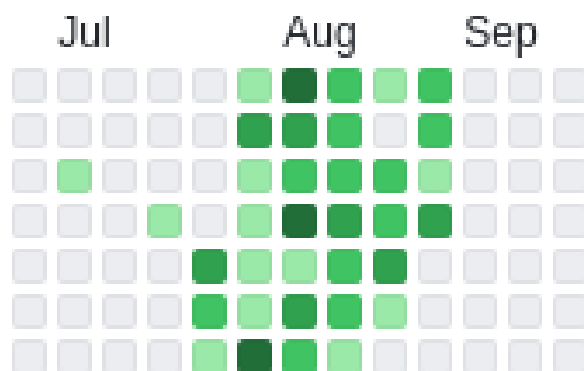
#### Introduzione

La Piattaforma per la Gestione degli Esami Universitari, d'ora in poi chiamata PiGEU per comodità, è una soluzione che permette di gestire uno scenario universitario in cui siano presenti insegnamenti e corsi di laurea, docenti, studenti e segretari, iscrizioni e verbalizzazioni ai diversi appelli di esame di ogni insegnamento, generazione di documentazioni valide per gli studenti quali i certificati di carriera, visibili o anche scaricabili in formato PDF.

PiGEU, nel suo complesso, abbraccia principalmente quattro tecnologie, quali:

- **PostgreSQL** per la gestione del database nel quale tramite *tabelle* vi è la memorizzazione dei dati di tutte le entità coinvolte, tramite *trigger* vengono controllate alcune proprietà della base di dati per preservare la consistenza dell'informazione e rispettare le specifiche fornite
- **HTML** come linguaggio di markup per rendere l'accesso ai dati maggiormente fruibile e comprensibile da parte dell'utente, rappresentando i dati in forma testuale o grafica (tabelle) e consentendo la possibilità di interrogare la base di dati mediante bottoni. Questo linguaggio è usato per rendere PiGEU maggiormente *user-friendly*
- **PHP** linguaggio di scripting server-side che "abbraccia" le due tecnologie sopra indicate, utilizzato per:
  - eseguire in linguaggio SQL-DML le interrogazioni al database che vengono fatte da parte dell'utente tramite la compilazione di form in HTML, inviate tramite la pressione di bottoni HTML
  - realizzare documenti in PDF tramite l'apposita libreria TCPDF
- **Javascript**, linguaggio di scripting (usato in questo caso per la parte client-side) che è stato utilizzato per
  - eseguire chiamate AJAX migliorando la performance di navigazione
  - eseguire interrogazioni al database in maniera più semplice, senza la pressione di bottoni di submit ma solamente con la digitazione da tastiera ad esempio durante la ricerca di una utenza

L'intero sviluppo del progetto in tutte le sue fasi, è documentato in un apposito repository di Github al seguente link <https://github.com/ffont28/PiGEU> di cui qui sotto viene mostrata un'immagine indicante gli stati di commit



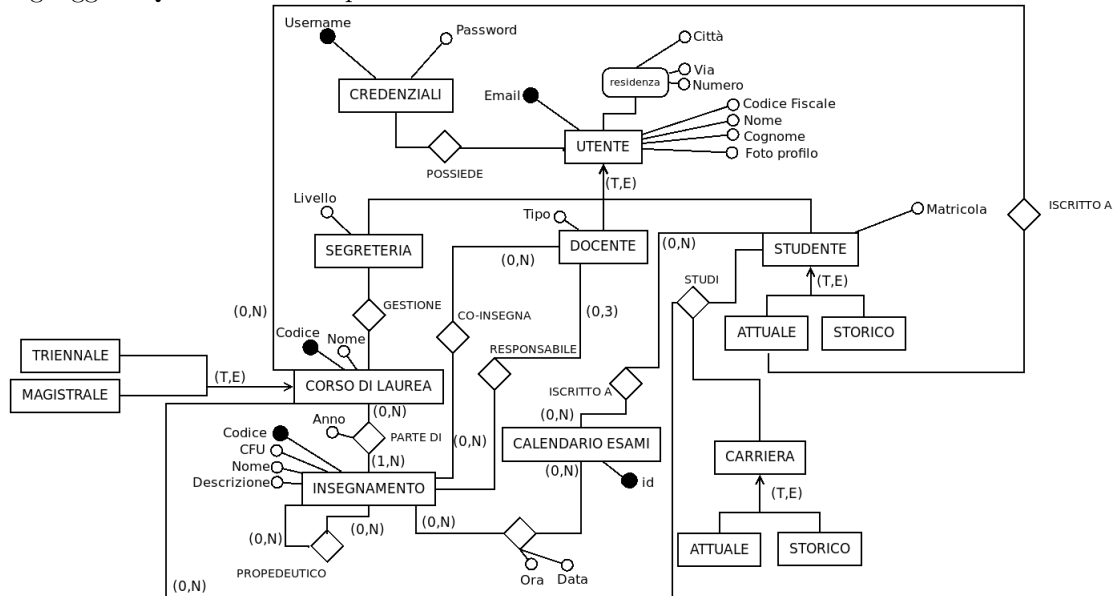
Inoltre la piattaforma è raggiungibile al seguente link: [www.pigeu.it:8081](http://www.pigeu.it:8081)

# Contents

<b>1</b>	<b>Schema concettuale (ER) della base di dati</b>	<b>4</b>
<b>2</b>	<b>Schema logico (relazionale) della base di dati</b>	<b>5</b>
<b>3</b>	<b>Esauriente descrizione delle funzioni realizzate</b>	<b>13</b>
3.1	Funzioni SQL-DML Richieste . . . . .	13
3.1.1	Rimozione di uno studente - [requisito 2.2.1] . . . . .	13
3.1.2	Correttezza delle iscrizioni agli esami - [requisito 2.2.2] . . . . .	14
3.1.3	Correttezza del calendario d'esame - [requisito 2.2.3] . . . . .	16
3.1.4	Produzione della carriera completa di uno studente - [requisito 2.2.4] . . . . .	17
3.1.5	Produzione della carriera valida di uno studente - [requisito 2.2.5] . . . . .	19
3.1.6	Produzione delle informazioni su un corso di laurea - [requisito 2.2.6] . . . . .	21
3.1.7	Il docente responsabile . . . . .	21
3.2	Ulteriori Funzioni realizzate . . . . .	22
3.2.1	Il loop di propedeuticità . . . . .	22
3.2.2	Carriera completa con tutti gli insegnamenti . . . . .	23
3.2.3	I tempi della propedeuticità . . . . .	24
3.2.4	La data dell'esame . . . . .	25
3.2.5	Disiscriversi da un esame già verbalizzato . . . . .	26
<b>4</b>	<b>Prove di funzionamento</b>	<b>27</b>
4.1	L'inserimento di un utente . . . . .	27
4.1.1	Studente . . . . .	27
4.1.2	Docente . . . . .	29
4.2	L'inserimento di un nuovo insegnamento . . . . .	30
4.3	L'iscrizione a un esame . . . . .	31
4.3.1	Iscrizione all'esame di un insegnamento al di fuori del corso di laurea . . . . .	31
4.3.2	Iscrizione senza rispettare la propedeuticità . . . . .	33
4.4	Calendarizzare un esame . . . . .	34
4.5	Il docente Responsabile . . . . .	36
<b>5</b>	<b>Funzionalità <i>user-friendly</i></b>	<b>39</b>
5.1	Icona di connessione alla base di dati . . . . .	39
5.2	Recupero delle credenziali . . . . .	39
5.3	Produzione dei documenti in PDF . . . . .	39
<b>6</b>	<b>Sicurezza di PiGEU</b>	<b>39</b>
6.1	crittografia delle credenziali . . . . .	39
6.2	autorizzazioni per determinati utenti . . . . .	39
6.2.1	la navbar limitata all'utente . . . . .	40
6.2.2	il controllo in homepage del tipo di utente . . . . .	40
6.2.3	controllo in testa a ogni pagina . . . . .	40
6.2.4	Prevenzione attacchi DOS . . . . .	40
6.2.5	Modifica password iniziale . . . . .	40

# 1 Schema concettuale (ER) della base di dati

Di seguito lo schema concettuale alla base della progettazione della base di dati. Lo schema seguente è stato successivamente ristrutturato per la realizzazione dello schema logico, dato che la relazione in linguaggio SQL-DDL viene espressa con una tabella.



## 2 Schema logico (relazionale) della base di dati

Di seguito viene fornito lo schema logico della base di dati di PiGEU, tramite i comandi DDL utilizzati. La base di dati contiene 20 tabelle: per ognuna di esse verrà fornito qualche commento per spiegare al meglio le scelte implementative derivanti dai requisiti assegnati

1. La tabella **utente** contiene i dati di qualsiasi utente, che sia di tipo *segreteria*, *studente* oppure *docente*. Questa soluzione di utenza generica è pensata appositamente nell'ottica della *scalabilità* di questo software, così da poter aggiungere anche altri tipi di utenze mediante ulteriori tabelle.

Questa tabella contiene i dati fondamentali di un utente, quali l'**email** che è l'identificatore univoco (chiave primaria) di ciascun utente nel database, corrisponde a quella che è l'email istituzionale di università, il suo **nome** e **cognome**, il suo **indirizzo** e **città** di residenza, il suo **codice fiscale** da cui si può desumere la *data di nascita* e il *luogo di nascita* e da ultimo l'**emailpersonale** che rappresenta la mail di recupero dell'utente, al di fuori del dominio email di università.

utente

```
1 CREATE TABLE public.utente (  
2     email character varying(50) NOT NULL,  
3     nome character varying(50),  
4     cognome character varying(50),  
5     indirizzo character varying(255),  
6     citta character varying(255),  
7     codicefiscale character varying(16),  
8     emailpersonale character varying(60),  
9     PRIMARY KEY (email)  
10 );
```

2. La tabella **credenziali** contiene per ciascun utente (denominato **username**), la corrispettiva **password** cifrata con l'algoritmo MD5. Il senso di questa tabella è, come è facile intuire, poter garantire l'accesso solamente agli utenti autorizzati.

credenziali

```
1 CREATE TABLE public.credenziali (  
2     username character varying(50) NOT NULL,  
3     password character varying(50),  
4     PRIMARY KEY (username),  
5     FOREIGN KEY (username) REFERENCES public.utente(email) ON UPDATE  
        CASCADE ON DELETE CASCADE  
6 );
```

3. La tabella **Segreteria** contiene i dati degli utenti di tipo *segreteria* identificati da un livello come ad esempio la segreteria del personale piuttosto che la segreteria studenti o segreteria didattica. L'attributo **utente** riferenzia l'email della tabella **Utente**

#### Segreteria

```
1 CREATE TABLE public.segreteria (  
2     utente character varying(50) NOT NULL,  
3     livello character varying(9) NOT NULL,  
4     PRIMARY KEY (utente),  
5     FOREIGN KEY (utente) REFERENCES public.utente(email) ON UPDATE CASCADE  
      ON DELETE CASCADE  
6 );
```

4. La tabella **Docente** contiene i dati degli utenti di tipo *docente*. Oltre a riferenziare l'email della tabella **Utente**, contiene anche un attributo **tipo** che indica il tipo di docente, ad esempio "ordinario", "associato", "a contratto",...

#### Docente

```
1 CREATE TABLE public.docente (  
2     utente character varying(50) NOT NULL,  
3     tipo character varying(50),  
4     PRIMARY KEY (utente),  
5     FOREIGN KEY (utente) REFERENCES public.utente(email) ON UPDATE CASCADE  
      ON DELETE CASCADE  
6 );
```

5. La tabella **Studente** contiene i dati degli utenti di tipo *studente*. Anch'essa riferenzia l'email della tabella **Utente**, e contiene due attributi caratteristici di uno studente:

- l'attributo **matricola** che indica la matricola dello studente
- l'attributo **corso\_di\_laurea** in riferimento al corso di laurea a cui risulta iscritto lo studente

#### Studente

```
1 CREATE TABLE public.studente (  
2     utente character varying(50) NOT NULL,  
3     matricola serial NOT NULL,  
4     corso_di_laurea character varying(5),  
5     PRIMARY KEY (utente),  
6     FOREIGN KEY (corso_di_laurea) REFERENCES public.corso_di_laurea(codice)  
      ON UPDATE CASCADE ON DELETE CASCADE,  
7     FOREIGN KEY (utente) REFERENCES public.utente(email) ON UPDATE CASCADE  
      ON DELETE CASCADE  
8 );
```

6. La tabella [Corso\\_di\\_laurea](#) contiene i dati di un corso di laurea, quali:

- il `codice`, identificatore univoco di uno specifico corso di laurea
- il `nome` del corso di laurea
- il `tipo` del corso di laurea, che può essere
  - *triennale*, ovvero della durata di tre anni
  - *magistrale* ovvero della durata di due anni
  - *magistrale a ciclo unico* della durata di cinque anni

Corso di Laurea

```
1 CREATE TABLE public.corso_di_laurea (  
2     codice character varying(5) NOT NULL,  
3     nome character varying(50) ,  
4     tipo character varying(50) ,  
5     PRIMARY KEY (codice)  
6 );
```

7. La tabella [Insegnamento](#) contiene i dati relativi a un insegnamento, quali:

- il `codice`, identificatore univoco di uno specifico insegnamento
- il `nome` dell'insegnamento
- una `descrizione` testuale dell'insegnamento, che enumera le conoscenze che tale disciplina vuole trasmettere allo studente
- i `cfu`, ovvero i crediti formativi universitari che l'insegnamento attribuisce allo studente che consegue un profitto in questa disciplina

❗ l'attributo indicante l'*anno* in cui è previsto l'insegnamento è volutamente omissso e verrà specificata la motivazione successivamente nella descrizione della tabella [insegnamento parte di cdl](#)

Insegnamento

```
1 CREATE TABLE public.insegnamento (  
2     codice character varying(10) NOT NULL,  
3     nome character varying(50) ,  
4     descrizione text ,  
5     cfu character(2) ,  
6     PRIMARY KEY (codice)  
7 );
```

8. Ogni insegnamento ha un [docente\\_responsabile](#) ovvero un docente che si occupa dell'organizzazione didattica dell'insegnamento come ad esempio, come vedremo in seguito, la pianificazione del calendario esami per l'insegnamento, e la verbalizzazione degli esiti degli esami dell'insegnamento. Questa tabella quindi indica la relazione tra un `docente` e un `insegnamento` i quali riferenziano le omonime tabelle nelle loro chiavi primarie.

#### Docente Responsabile

```
1 CREATE TABLE public.docente_responsabile (  
2     docente character varying(50) NOT NULL,  
3     insegnamento character varying(10) NOT NULL  
4     PRIMARY KEY (docente, insegnamento),  
5     FOREIGN KEY (docente) REFERENCES public.docente(utente) ON UPDATE  
        CASCADE ON DELETE CASCADE,  
6     FOREIGN KEY (insegnamento) REFERENCES public.insegnamento(codice) ON  
        UPDATE CASCADE ON DELETE CASCADE  
7 );
```

9. Ogni insegnamento può avere uno o più docenti: la tabella [insegna](#) descrive tale situazione. Questa tabella conterrà solamente i docenti di un insegnamento che non siano già responsabili di quell'insegnamento. Come nella tabella [docente\\_responsabile](#) i due attributi fanno riferimento a un docente e a un insegnamento.

#### Insegna

```
1 CREATE TABLE public.insegna (  
2     docente character varying(50) NOT NULL,  
3     insegnamento character varying(10) NOT NULL,  
4     PRIMARY KEY (docente, insegnamento),  
5     FOREIGN KEY (docente) REFERENCES public.docente(utente) ON UPDATE  
        CASCADE ON DELETE CASCADE,  
6     FOREIGN KEY (insegnamento) REFERENCES public.insegnamento(codice) ON  
        UPDATE CASCADE ON DELETE CASCADE  
7 );
```

10. La tabella [insegnamento\\_parte\\_di\\_cdl](#) descrive la relazione per la quale ogni insegnamento non sia a se stante, ma faccia parte di un determinato corso di laurea.

❗ Si noti che una scelta implementativa che diverge dalle specifiche è inserire l'attributo `anno` all'interno di questa relazione e non come proprietà di un singolo corso di laurea, dal momento che uno stesso insegnamento può essere erogato in più corsi di laurea in anni diversi (ad esempio l'insegnamento `i` può essere erogato per il corso di laurea `c1` al primo anno, mentre per il corso di laurea `c2` al terzo anno)

#### Insegnamento parte di un Corso di Laurea

```
1 CREATE TABLE public.insegnamento_parte_di_cdl (  
2     insegnamento character varying(10) NOT NULL,  
3     corso_di_laurea character varying(5) NOT NULL,  
4     anno integer NOT NULL,  
5     PRIMARY KEY (insegnamento, corso_di_laurea, anno),  
6     FOREIGN KEY (corso_di_laurea) REFERENCES public.corso_di_laurea(codice)  
        ON UPDATE CASCADE ON DELETE CASCADE,  
7     FOREIGN KEY (insegnamento) REFERENCES public.insegnamento(codice) ON  
        UPDATE CASCADE ON DELETE CASCADE  
8 );
```



11. La tabella `propedeuticita` descrive la relazione per la quale eventualmente un insegnamento `insegnamento1` possa essere propedeutico a un altro insegnamento `insegnamento2` all'interno di un `corso_di_laurea`. Anche qui è importante notare come un insegnamento sia propedeutico a un altro solamente nel contesto di un corso di laurea, non in maniera assoluta.

#### Propedeuticità

```
1 CREATE TABLE public.propedeuticita (  
2     insegnamento1 character varying(10) NOT NULL,  
3     insegnamento2 character varying(10) NOT NULL,  
4     corso_di_laurea character varying(5) NOT NULL,  
5     PRIMARY KEY (insegnamento1, insegnamento2, corso_di_laurea),  
6     FOREIGN KEY (insegnamento1) REFERENCES public.insegnamento(codice) ON  
        UPDATE CASCADE ON DELETE CASCADE,  
7     FOREIGN KEY (insegnamento2) REFERENCES public.insegnamento(codice) ON  
        UPDATE CASCADE ON DELETE CASCADE  
8     FOREIGN KEY (corso_di_laurea) REFERENCES public.corso_di_laurea(codice)  
        ON UPDATE CASCADE ON DELETE CASCADE  
9 );
```

12. La tabella `insegnamenti_per_carriera` esprime un sottile concetto presente nell'ambito universitario: all'atto dell'immatricolazione a uno studente viene attribuito un *piano di studi* che è rappresentato proprio da questa tabella: a ogni studente viene assegnato un elenco di insegnamenti da superare. Se successivamente all'immatricolazione dello studente (che qui coincide con l'atto dell'inserimento nel database) in un dato corso di laurea viene inserito un nuovo insegnamento, non viene modificato il piano di studi dello studente già iscritto precedentemente la modifica. L'attributo `timestamp` serve per tenere una marca temporale dell'atto della realizzazione del presente piano di insegnamenti da superare. Un'ulteriore indicazione di quanto detto finora è dato dal **vincolo di chiave esterna** che referencia solamente l'attributo `utente` della tabella `studente` (e non anche il `codice` la tabella `insegnamento`!): se lo studente `s` viene rimosso dalla tabella `studente` ad esempio perchè abbandona gli studi oppure si è laureato (in questo caso i dati verranno spostati in apposite tabelle di *storico* che verranno mostrate in seguito), decade l'insieme di insegnamenti di cui superare gli esami ai fini della carriera. Qualora `s` volesse reinscrivere, si troverebbe assegnati gli insegnamenti di cui conseguire profitto all'atto della nuova iscrizione.

#### Insegnamenti per carriera

```
1 CREATE TABLE public.insegnamenti_per_carriera (  
2     studente character varying(50) NOT NULL,  
3     insegnamento character varying(10) NOT NULL,  
4     "timestamp" timestamp(6) without time zone,  
5     PRIMARY KEY (studente, insegnamento),  
6     FOREIGN KEY (studente) REFERENCES public.studente(utente) ON UPDATE  
        CASCADE ON DELETE CASCADE  
7 );
```

13. La tabella `calendario_esami` rappresenta un calendario di esami in cui vengono indicati:

- un `insegnamento` che riferenzia l'attributo `codice` della tabella `insegnamento`
- la data dell'esame
- l'ora dell'esame
- un `id` univoco identificativo di uno specifico *esame* che è costituito dalla tripla sopra indicata, ovvero  $\langle insegnamento, data, ora \rangle$

#### Calendario Esami

```
1 CREATE TABLE public.calendario_esami (  
2     insegnamento character varying(10) NOT NULL,  
3     data date NOT NULL,  
4     ora time without time zone NOT NULL,  
5     id serial PRIMARY KEY,  
6     FOREIGN KEY (insegnamento) REFERENCES public.insegnamento(codice) ON  
        UPDATE CASCADE ON DELETE CASCADE  
7 );
```

14. La tabella `iscrizione` esprime l'iscrizione di uno `studente` a un `esame`, quest'ultimo attributo che riferenzia la tabella `calendario_esami` nel suo attributo `id`

#### Iscrizione

```
1 CREATE TABLE public.iscrizione (  
2     studente character varying(50) NOT NULL,  
3     esame integer NOT NULL,  
4     PRIMARY KEY (studente, esame),  
5     FOREIGN KEY (studente) REFERENCES public.studente(utente) ON UPDATE  
        CASCADE ON DELETE CASCADE,  
6     FOREIGN KEY (esame) REFERENCES public.calendario_esami(id) ON UPDATE  
        CASCADE ON DELETE CASCADE  
7 );
```

15. La tabella `Carriera` contiene i dati della carriera di ciascun studente, ovvero per ogni `insegnamento` di cui sia stato sostenuto *almeno* un esame, la corrispettiva `valutazione` conseguita in tal data. Si noti l'enfasi sul termine "almeno" per indicare che in carriera vengono registrati tutti i tentativi di sostenimento di esame, essendo stabilita come chiave primaria la tripla  $\langle studente, insegnamento, data \rangle$  è costituita quella che in gergo viene detta *serie storica*.

#### Carriera

```
1 CREATE TABLE public.carriera (  
2     studente character varying(50) NOT NULL,  
3     insegnamento character varying(10) NOT NULL,  
4     valutazione smallint,  
5     data date NOT NULL,  
6     PRIMARY KEY (studente, data, insegnamento),  
7     FOREIGN KEY (studente, insegnamento) REFERENCES  
        public.insegnamenti_per_carriera(studente, insegnamento) ON UPDATE  
        CASCADE ON DELETE CASCADE  
8 );
```

16. La tabella `utente_storico` contiene gli utenti inseriti in storico. Per il momento l'unica tipologia di utenti per cui è previsto lo storico sono gli *studenti*. La struttura e la semantica è del tutto identica alla tabella `utente`

#### Utente Storico

```
1 CREATE TABLE public.utente_storico (  
2     email character varying(50) NOT NULL,  
3     nome character varying(50),  
4     cognome character varying(50),  
5     indirizzo character varying(255),  
6     citta character varying(255),  
7     codicefiscale character varying(16),  
8     emailpersonale character varying(60),  
9     PRIMARY KEY (email)  
10 );
```

17. La tabella `studente_storico` contiene i dati degli studenti inseriti in storico. La struttura e la semantica è del tutto identica alla tabella `studente`

#### Studente Storico

```
1 CREATE TABLE public.studente_storico (  
2     utente character varying(50) NOT NULL,  
3     matricola integer,  
4     corso_di_laurea character varying(5),  
5     PRIMARY KEY (utente),  
6     FOREIGN KEY (utente) REFERENCES public.utente_storico(email) ON UPDATE  
    CASCADE ON DELETE CASCADE  
7 );
```

18. La tabella `carriera_storico` contiene i dati della carriera di studenti inseriti in storico. La struttura e la semantica è del tutto identica alla tabella `carriera`

#### Carriera Storico

```
1 CREATE TABLE public.carriera_storico (  
2     studente character varying(50) NOT NULL,  
3     insegnamento character varying(10) NOT NULL,  
4     valutazione smallint,  
5     data date NOT NULL,  
6     PRIMARY KEY (studente, insegnamento, data),  
7     FOREIGN KEY (studente) REFERENCES public.studente_storico(utente) ON  
    UPDATE CASCADE ON DELETE CASCADE  
8 );
```

19. La tabella `foto_profilo` contiene, per ciascun `utente`, il corrispettivo percorso interno al server in cui si trova l'immagine del profilo, tramite l'attributo `path`. Inoltre l'attributo `timestamp` serve per tenere memoria delle vecchie foto profilo dell'utente in questione.

foto\_profilo

```
1 CREATE TABLE public.foto_profilo (  
2     utente character varying(50) NOT NULL,  
3     path character varying(200),  
4     "timestamp" timestamp(6) without time zone NOT NULL,  
5     PRIMARY KEY (utente, "timestamp"),  
6     FOREIGN KEY (utente) REFERENCES public.utente(email) ON UPDATE CASCADE  
      ON DELETE CASCADE  
7 );
```

20. La tabella `recupero_credenziali` contiene, per ciascun `utente` che abbia fatto richiesta di recupero credenziali, l'indirizzo email istituzionale dell'utente e un token assegnato all'utente stesso, che verrà inviato tramite email all'indirizzo di recupero come parametro di una richiesta GET, per ripristinare la password. Il token verrà rimosso dalla tabella nel momento in cui dall'email viene cliccato sul link generato per il recupero.

recupero\_credenziali

```
1 CREATE TABLE public.recupero_credenziali (  
2     utente character varying(50) NOT NULL,  
3     randomvalue character varying(50) NOT NULL,  
4     PRIMARY KEY (utente, randomvalue),  
5     FOREIGN KEY (utente) REFERENCES public.utente(email) ON UPDATE CASCADE  
      ON DELETE CASCADE  
6 );
```

## 3 Esauriente descrizione delle funzioni realizzate

### 3.1 Funzioni SQL-DML Richieste

Questa sezione ha lo scopo di mostrare le funzioni in linguaggio SQL per la manipolazione dei dati interni al lato PostgreSQL di PiGEU. Per ciascuna funzione verrà mostrata l'utilità ed eventuali osservazioni.

#### 3.1.1 Rimozione di uno studente - [requisito 2.2.1]

Nelle specifiche indicate, la rimozione di uno studente **s** non corrisponde con il solo comando di **DELETE** all'interno della tabella **utente** ma comporta il trasferimento di tutte le informazioni relative ad **s** in apposite tabelle di *storico*. La funzione **sposta\_dati\_studente** contiene diversi comandi DML che consentono di soddisfare il requisito in questione. Di seguito il codice sorgente della funzione che oltre alle operazioni richieste, restituisce anche un messaggio di stato oppure, nel caso di sollevamento di eccezioni, il messaggio di eccezione.

sposta\_dati\_studente(E varchar)

```
1 CREATE OR REPLACE FUNCTION sposta_dati_studente(E varchar) RETURNS TEXT AS $$
2 DECLARE
3     status TEXT := 'Dati spostati con successo.';
4 BEGIN
5     -- INSERT utente --> utente_storico
6     INSERT INTO utente_storico (email, nome, cognome, indirizzo, citta,
7                                codicefiscale, emailpersonale)
8     SELECT email, nome, cognome, indirizzo, citta, codicefiscale, emailpersonale
9     FROM utente where email = $1;
10
11    -- INSERT studente --> studente_storico
12    INSERT INTO studente_storico (utente, matricola, corso_di_laurea)
13    SELECT utente, matricola, corso_di_laurea
14    FROM studente WHERE utente = $1;
15
16    -- INSERT carriera --> carriera_storico
17    INSERT INTO carriera_storico (studente, insegnamento, valutazione, data)
18    SELECT studente, insegnamento, valutazione, data
19    FROM carriera WHERE studente = $1;
20
21    -- cancello i dati dalla tabella utente dopo averli spostati in utente_storico
22    DELETE FROM utente where email = $1;
23    -- le seguenti operazioni sono fatte gia' in CASCADE
24    -- cancello i dati dalla tabella studente dopo averli spostati in
25    --   stutente_storico
26    -- DELETE FROM studente WHERE utente = $1;
27    -- cancello i dati dalla tabella carriera dopo averli spostati in
28    --   carriera_storico
29    -- DELETE FROM carriera WHERE email = $1;
30    -- cancello i dati dalla tabella insegnamenti_per_carriera
31    -- DELETE FROM insegnamenti_per_carriera WHERE studente = $1;
32    RETURN status;
33    -- Gestione delle eccezioni
34 EXCEPTION
35     WHEN OTHERS THEN
36         -- In caso di errore restituisco il messaggio di errore
37         status := 'Errore: ' || SQLERRM;
```

```

35     RETURN status;
36 END;
37 $$ LANGUAGE plpgsql;

```

### 3.1.2 Correttezza delle iscrizioni agli esami - [requisito 2.2.2]

Nel *requisito 2.2.2* viene detto che

Ciascuno studente può iscriversi ad un esame solo se l'insegnamento è previsto dal proprio corso di laurea, e solamente se tutte le propedeuticità sono rispettate.

- La prima condizione, cioè la possibilità di iscrizione ai soli esami di insegnamenti del proprio corso di laurea, viene controllata tramite un trigger denominato `no.iscriz.se.non.in.CdL` di cui viene fornito il codice sorgente

`no.iscriz.se.non.in.CdL`

```

1 CREATE OR REPLACE TRIGGER no.iscriz.se.non.in.CdL
2   BEFORE INSERT OR UPDATE ON iscrizione
3   FOR EACH ROW
4   EXECUTE FUNCTION check_appartenenza_cdl();

```

Tale trigger prima che venga inserita una riga nella tabella `iscrizione` chiama l'esecuzione della funzione `check_appartenenza_cdl` di seguito mostrata

`check_appartenenza_cdl()`

```

1 CREATE OR REPLACE FUNCTION check_appartenenza_cdl() RETURNS TRIGGER as $$
2 BEGIN
3     PERFORM * FROM calendario_esami c
4         INNER JOIN insegnamenti_per_carriera ipc ON ipc.insegnamento =
5             c.insegnamento
6         WHERE ipc.studente = NEW.studente AND NEW.esame = c.id;
7     IF FOUND THEN
8         -- l'esame che si vuole inserire in calendario_esami e'' presente in
9         -- ipc
10        RETURN NEW;
11    ELSE
12        -- l'esame che si vuole inserire in calendario_esami NON e''
13        -- presente in ipc
14        RAISE NOTICE 'ATTENZIONE: non e'' consentita l''iscrizione ad un esame
15        che non appartiene al corso di laurea di uno studente';
16        PERFORM pg_notify('notifica', 'ATTENZIONE: non e'' consentita
17        l''iscrizione ad un esame che non appartiene al corso di laurea di
18        uno studente');
19        RETURN NULL;
20    END IF;
21 END;
22 $$ language 'plpgsql';

```

La funzione appena mostrata mostra alcune particolarità dovute all'implementazione scelta:

- a livello di progettazione, si verifica non che l'insegnamento a cui si vuole iscrivere lo studente appartenga non tanto al corso di laurea, quanto all'*elenco di insegnamenti*, presente nella tabella `insegnamenti_per_carriera` che lo studente è tenuto a superare, che

viene generato all'atto dell'immatricolazione dello studente. Tale elenco di insegnamenti viene preso riferendosi al corso di laurea quindi si aderisce alle specifiche date, pur aggiungendo una sottigliezza, ovvero quella della mutabilità di un corso di laurea che oggi può contenere  $n$  insegnamenti, domani ne può contenere  $m$  con  $n \neq m$

- a livello di funzionalità, la funzione contiene anche i messaggi di informazione:
  - \* la **NOTICE** verrà mostrata come messaggio se la funzione viene chiamata da terminale
  - \* il comando **PERFORM pg\_notify** invece genererà una notifica che potrà essere catturata dal linguaggio di scripting PHP
- La seconda condizione invece, ovvero il rispetto delle propedeuticità, è controllata dal seguente trigger:

#### no\_esami\_senza\_propedeuticità

```
1 CREATE OR REPLACE TRIGGER no_esami_senza_propedeuticità
2   BEFORE INSERT OR UPDATE ON iscrizione
3   FOR EACH ROW
4   EXECUTE FUNCTION check_propedeuticità();
```

che chiama l'esecuzione della seguente funzione

#### check\_propedeuticità()

```
1 CREATE OR REPLACE FUNCTION check_propedeuticità() RETURNS TRIGGER as $$
2 BEGIN
3   -- cerco se c'è in propedeuticità e nel caso che in carriera
4   -- 'insegnamento1' sia superato
5   PERFORM * FROM propedeuticità p
6   -- INNER JOIN carriera c ON c.insegnamento = p.insegnamento1 AND
7   -- c.studente = NEW.studente
8   INNER JOIN calendario_esami ce ON ce.insegnamento = p.insegnamento2
9   INNER JOIN studente s ON s.corso_di_laurea = p.corso_di_laurea
10  WHERE ce.id = NEW.esame AND s.utente = NEW.studente;
11  IF NOT FOUND THEN
12    RETURN NEW;
13  ELSE
14    PERFORM * FROM carriera c
15    INNER JOIN calendario_esami ce ON ce.insegnamento =
16    c.insegnamento
17    INNER JOIN propedeuticità p ON p.insegnamento1 =
18    ce.insegnamento
19    INNER JOIN studente s ON s.corso_di_laurea =
20    p.corso_di_laurea
21    WHERE c.studente = NEW.studente
22    AND c.valutazione >= 18;
23    IF FOUND THEN
24      RETURN NEW;
25    ELSE
26      RAISE NOTICE 'ATTENZIONE: non è possibile iscriversi
27      ad un esame senza aver superato la sua
28      propedeuticità';
29      PERFORM pg_notify('notifica', 'ATTENZIONE: non è
30      possibile iscriversi ad un esame senza aver
31      superato la sua propedeuticità');
32      RETURN NULL;
33    END IF;
```

```

24         END IF;
25     END;
26 $$ language 'plpgsql';

```

Le osservazioni su quanto riguarda il sollevamento della **NOTICE** che verrà catturata poi dallo script PHP grazie al comando **PERFORM pg\_notify** sono del tutto analoghe al punto precedente.

### 3.1.3 Correttezza del calendario d'esame - [requisito 2.2.3]

Viene richiesto, tra i requisiti:

Non è possibile programmare, nella stessa giornata, appelli per più esami dello stesso anno di un corso di laurea

Questa specifica trova una applicazione particolare nel realizzare il trigger che mantenga lo stato di correttezza nella base di dati:

no\_esami\_stesso\_anno\_stesso\_giorno

```

1 CREATE OR REPLACE TRIGGER no_esami_stesso_anno_stesso_giorno
2 BEFORE INSERT OR UPDATE ON calendario_esami
3 FOR EACH ROW
4 EXECUTE FUNCTION check_inserimento_esame();

```

Dal momento che per scelta implementativa l'attributo anno non fa parte di un **insegnamento** ma è identificabile solamente quando un **insegnamento** è parte di un **corso\_di\_laurea**, informazione presente nella tabella **insegnamento\_parte\_di\_cdl**, la funzione che effettua l'interrogazione alla base di dati risulterà leggermente articolata. Di seguito il codice sorgente:

check\_inserimento\_esame()

```

1 CREATE OR REPLACE FUNCTION check_inserimento_esame() RETURNS TRIGGER as $$
2 BEGIN
3     WITH esami presenti AS (SELECT DISTINCT cl.insegnamento,
4         ip.corso_di_laurea, ip.anno, cl.data
5         FROM calendario_esami cl
6         INNER JOIN insegnamento_parte_di_cdl ip ON
7             cl.insegnamento = ip.insegnamento),
8     cdltarget AS (SELECT ip.corso_di_laurea, ip.anno
9         FROM insegnamento_parte_di_cdl ip
10        WHERE ip.insegnamento = NEW.insegnamento)
11     PERFORM *
12     FROM esami presenti e INNER JOIN cdltarget c
13     ON e.corso_di_laurea = c.corso_di_laurea AND e.anno = c.anno
14     WHERE e.data = NEW.data;
15 IF FOUND THEN
16     RAISE NOTICE 'ATTENZIONE: e'' gia'' presente un altro esame dello stesso
17     anno per la data selezionata';
18     PERFORM pg_notify('notifica', 'ATTENZIONE: e'' gia'' presente un altro
19     esame dello stesso anno per la data selezionata');
20     RETURN NULL;
21 ELSE
22     RETURN NEW;
23 END IF;

```



```

20 END;
21 $$ language 'plpgsql';

```

### 3.1.4 Produzione della carriera completa di uno studente - [requisito 2.2.4]

Ogni università deve potere produrre, come documento ufficiale istituzionale, la carriera dei propri studenti, attestante gli studi effettuati. Definiamo con **carriera completa** l'insieme di insegnamenti di cui lo studente abbia sostenuto almeno una volta l'esame, includendo quindi esami anche ripetuti ed esami non superati. Il *requisito 2.2.4* chiede che:

- ciascuno studente possa produrre la propria *carriera completa*
- la segreteria possa produrre la *carriera completa* di ciascuno studente

in risposta a questa richiesta vi è la funzione **carriera\_completa** di cui viene mostrato il codice sorgente

```

                                carriera_completa(TARGET varchar)

1 CREATE OR REPLACE FUNCTION carriera_completa(TARGET varchar) RETURNS TABLE (
2     studente varchar,
3     nomstu varchar,
4     cogstu varchar,
5     cdl varchar,
6     matr integer,
7     codins varchar,
8     nomins varchar,
9     nomdoc varchar,
10    cogdoc varchar,
11    voto varchar,
12    data date
13 ) AS $$
14 BEGIN
15     RETURN QUERY
16     SELECT DISTINCT ic.studiante,
17                    u2.nome nomstu,
18                    u2.cognome cogstu,
19                    cdl.nome cdl,
20                    s.matricola matr,
21                    ic.insegnamento,
22                    i.nome,
23                    u.nome nomedoc,
24                    u.cognome cogndoc,
25                    CASE
26                        WHEN c.valutazione = 31 THEN '30 e LODE'
27                        ELSE c.valutazione::VARCHAR
28                    END,
29                    c.data
30     FROM insegnamenti_per_carriera ic
31          INNER JOIN carriera c ON ic.insegnamento = c.insegnamento
32          INNER JOIN docente_responsabile d ON d.insegnamento =
33              ic.insegnamento
34          INNER JOIN utente u ON d.docente = u.email
35          INNER JOIN utente u2 ON ic.studiante = u2.email
36          INNER JOIN studente s ON ic.studiante = s.utente
          INNER JOIN corso_di_laurea cdl ON s.corso_di_laurea = cdl.codice

```

```

37         INNER JOIN insegnamento i ON ic.insegnamento = i.codice
38     WHERE ic.studente = TARGET AND c.studente = TARGET
39 END;
40 $$ LANGUAGE plpgsql;

```

La funzione nella sua segnatura presenta il parametro formale **E varchar** che nel caso dello studente, sarà aggiornato dall'username dello studente ovvero il suo indirizzo email, mentre nel caso in cui la funzione venga invocata dalla segreteria, il parametro **E** verrà aggiornato con il valore dell'attributo **username** dello studente di cui la segreteria vuole produrre la documentazione di carriera completa.

Discorso analogo nel caso in cui la segreteria debba produrre la *carriera completa* di uno **studente storico**:

```

                                carriera_completa_sto(TARGET varchar)

1  CREATE OR REPLACE FUNCTION carriera_completa_sto(TARGET varchar) RETURNS TABLE (
2      studente varchar,
3      nomstu varchar,
4      cogstu varchar,
5      cdl varchar,
6      matr integer,
7      codins varchar,
8      nomins varchar,
9      nomdoc varchar,
10     cogdoc varchar,
11     voto varchar,
12     data date
13 ) AS $$
14 BEGIN
15     RETURN QUERY
16         SELECT DISTINCT c.studente,
17                        u2.nome nomstu,
18                        u2.cognome cogstu,
19                        cdl.nome cdl,
20                        s.matricola matr,
21                        c.insegnamento,
22                        i.nome,
23                        u.nome nomedoc,
24                        u.cognome cogndoc,
25                        CASE
26                            WHEN c.valutazione = 31 THEN '30 e LODE'
27                            ELSE c.valutazione::VARCHAR
28                        END,
29                        c.data
30     FROM carriera_storico c
31         INNER JOIN docente_responsabile d ON d.insegnamento =
32                                           c.insegnamento
33         INNER JOIN utente u ON d.docente = u.email
34         INNER JOIN utente_storico u2 ON c.studente = u2.email
35         INNER JOIN studente_storico s ON c.studente = s.utente
36         INNER JOIN corso_di_laurea cdl ON s.corso_di_laurea = cdl.codice
37         INNER JOIN insegnamento i ON c.insegnamento = i.codice
38     WHERE c.studente = TARGET;
39 END;
40 $$ LANGUAGE plpgsql;

```

### 3.1.5 Produzione della carriera valida di uno studente - [requisito 2.2.5]

Definiamo ora con **carriera valida** l'insieme di insegnamenti di cui lo studente abbia sostenuto almeno una volta l'esame, e che quest'ultimo sia stato superato (valutazione  $\geq 18$ ). In caso di esami sostenuti più volte, è da considerarsi solo la valutazione ottenuta in data più recente. Il *requisito 2.2.4* chiede che:

- ciascuno studente possa produrre la propria *carriera valida*
- la segreteria possa produrre la *carriera valida* di ciascuno studente

in risposta a questa richiesta questa volta vi è la funzione **carriera\_valida** di cui viene mostrato il codice sorgente

```
carriera_valida(TARGET varchar)

1 CREATE OR REPLACE FUNCTION carriera_valida(TARGET varchar) RETURNS TABLE (
2     studente varchar ,
3     nomstu varchar ,
4     cogstu varchar ,
5     cdl varchar ,
6     matr integer ,
7     codins varchar ,
8     nomins varchar ,
9     nomdoc varchar ,
10    cogdoc varchar ,
11    voto varchar ,
12    data date
13 ) AS $$
14 BEGIN
15     RETURN QUERY
16     SELECT DISTINCT ic.studente ,
17         u2.nome nomstu ,
18         u2.cognome cogstu ,
19         cdl.nome cdl ,
20         s.matricola matr ,
21         ic.insegnamento ,
22         i.nome ,
23         u.nome nomedoc ,
24         u.cognome cogndoc ,
25         CASE
26             WHEN c.valutazione = 31 THEN '30 e LODE'
27             ELSE c.valutazione::VARCHAR
28         END,
29         c.data
30 FROM insegnamenti_per_carriera ic
31     INNER JOIN carriera c ON ic.insegnamento = c.insegnamento
32     INNER JOIN docente_responsabile d ON d.insegnamento =
33         ic.insegnamento
34     INNER JOIN utente u ON d.docente = u.email
35     INNER JOIN utente u2 ON ic.studente = u2.email
36     INNER JOIN studente s ON ic.studente = s.utente
37     INNER JOIN corso_di_laurea cdl ON s.corso_di_laurea = cdl.codice
38     INNER JOIN insegnamento i ON ic.insegnamento = i.codice
39 WHERE ic.studente = TARGET AND c.studente = TARGET
40     AND c.valutazione >= 18
41     AND c.data = (SELECT MAX(c2.data)
```

```

41          FROM carriera c2 WHERE c2.insegnamento =
              ic.insegnamento);
42 END;
43 $$ LANGUAGE plpgsql;

```

La funzione nella sua segnatura presenta il parametro formale `E varchar` che nel caso dello studente, sarà aggiornato dall'`username` dello studente ovvero il suo indirizzo email, mentre nel caso in cui la funzione venga invocata dalla segreteria, il parametro `E` verrà aggiornato con il valore dell'attributo `username` dello studente di cui la segreteria vuole produrre la documentazione di carriera valida.

Discorso del tutto analogo nel caso in cui la segreteria debba produrre la *carriera valida* di uno **studente storico**:

```

                                carriera_valida_sto(TARGET varchar)
1 CREATE OR REPLACE FUNCTION carriera_valida_sto(TARGET varchar) RETURNS TABLE (
2     studente varchar,
3     nomstu varchar,
4     cogstu varchar,
5     cdl varchar,
6     matr integer,
7     codins varchar,
8     nomins varchar,
9     nomdoc varchar,
10    cogdoc varchar,
11    voto varchar,
12    data date ) AS $$
13 BEGIN
14     RETURN QUERY
15     SELECT DISTINCT c.studente,
16                    u2.nome nomstu,
17                    u2.cognome cogstu,
18                    cdl.nome cdl,
19                    s.matricola matr,
20                    c.insegnamento,
21                    i.nome,
22                    u.nome nomedoc,
23                    u.cognome cogndoc,
24                    CASE
25                        WHEN c.valutazione = 31 THEN '30 e LODE'
26                        ELSE c.valutazione::VARCHAR
27                    END,
28                    c.data
29 FROM   carriera_storico c
30        INNER JOIN docente_responsabile d ON d.insegnamento =
              c.insegnamento
31        INNER JOIN utente u ON d.docente = u.email
32        INNER JOIN utente_storico u2 ON c.studente = u2.email
33        INNER JOIN studente_storico s ON c.studente = s.utente
34        INNER JOIN corso_di_laurea cdl ON s.corso_di_laurea = cdl.codice
35        INNER JOIN insegnamento i ON c.insegnamento = i.codice
36 WHERE  c.studente = TARGET
37        AND c.valutazione >= 18
38        AND c.data = (SELECT MAX(c2.data)
39                      FROM   carriera_storico c2 WHERE c2.insegnamento =
                          c.insegnamento);
40 END;
41 $$ LANGUAGE plpgsql;

```

### 3.1.6 Produzione delle informazioni su un corso di laurea - [requisito 2.2.6]

Ciascuno studente deve poter conoscere, per ogni corso di laurea, gli insegnamenti erogati con le rispettive descrizioni e docente responsabile. Per aderire a questo requisito, vi è la vista `informazioni_CdL` che viene mostrata di seguito:

`informazioni_CdL`

```
1 CREATE or replace VIEW informazioni_CdL AS
2 SELECT c.codice, c.nome, c.tipo, i.codice codicec, i.nome nomec, ip.anno ,
   i.descrizione, i.cfu, u.nome nomedoc, u.cognome cognomedoc
3 FROM corso_di_laurea c INNER JOIN insegnamento_parte_di_cdl ip ON c.codice =
   ip.corso_di_laurea
4         INNER JOIN insegnamento i ON ip.insegnamento = i.codice
5         INNER JOIN docente_responsabile d ON i.codice =
   d.insegnamento
6         INNER JOIN utente u ON u.email = d.docente;
```

### 3.1.7 Il docente responsabile

Viene richiesto, tra le specifiche, che un docente sia responsabile di al più tre insegnamenti: per aderire a questo requisito vi è il seguente trigger:

`responsab_non_piu_di_tre`

```
1 CREATE OR REPLACE TRIGGER responsab_non_piu_di_tre
2 BEFORE INSERT OR UPDATE ON docente_responsabile
3 FOR EACH ROW
4 EXECUTE FUNCTION check_docente_responsabile_max_tre();
```

che invoca, prima di inserire un docente come responsabile nella tabella `docente_responsabile`, la seguente funzione

`check_docente_responsabile_max_tre()`

```
1 CREATE OR REPLACE FUNCTION check_docente_responsabile_max_tre() RETURNS TRIGGER
   AS $$
2 DECLARE
3     nir INT;
4 BEGIN
5     SELECT COUNT(*)
6     INTO nir
7     FROM docente_responsabile dr
8     WHERE dr.docente = NEW.docente;
9
10    IF nir > 2 THEN
11        RAISE NOTICE 'ATTENZIONE: il docente e'' gia'' responsabile di tre
            insegnamenti';
12        PERFORM pg_notify('notifica', 'ATTENZIONE: il docente e'' gia''
            responsabile di tre insegnamenti');
13        RETURN NULL;
14    ELSE
15        RETURN NEW;
16    END IF;
17 END;
18 $$ LANGUAGE plpgsql;
```

## 3.2 Ulteriori Funzioni realizzate

### 3.2.1 Il loop di propedeuticità

Ricodiamo che nel dominio applicativo di PiGEU intendiamo che un insegnamento A si dice *propedeutico* ad un altro insegnamento B in un corso di laurea L se per iscriversi ad un esame dell'insegnamento B è obbligatorio aver superato con profitto (ovvero valutazione  $\geq 18$ ) l'esame dell'insegnamento A. Può capitare una situazione erronea in cui, dopo aver impostato lato segreteria un insegnamento A come propedeutico a B si inserisca anche la propedeuticità di B per A: questa situazione genera un circolo vizioso senza fine in cui non ci si potrà mai iscrivere a nessuno dei due esami. Analogo discorso se vi è una catena di propedeuticità il cui insegnamento a capo della catena è lo stesso della coda della catena. Per questa motivazione nella base di dati è presente il seguente trigger:

#### no\_cicli\_propedeuticità

```
1 CREATE OR REPLACE TRIGGER no_cicli_propedeuticità
2 BEFORE INSERT OR UPDATE ON propedeuticità
3 FOR EACH ROW
4 EXECUTE FUNCTION check_propedeuticità_ciclo();
```

che controlla che nell'inserimento di una propedeuticità non si vadano a formare cicli di propedeuticità come quello appena descritto. Tale trigger richiama la seguente funzione

#### check\_propedeuticità\_ciclo()

```
1 CREATE OR REPLACE FUNCTION check_propedeuticità_ciclo() RETURNS TRIGGER AS $$
2 DECLARE
3     counter INTEGER := 0;
4     size INTEGER;
5     instemp VARCHAR;
6     riga record;
7 BEGIN
8
9     -- tabella temporanea con le propedeuticità '' visitate
10    CREATE TEMP TABLE propedeuticità_visitate
11    (ins1 varchar, ins2 varchar, cdl varchar) ON COMMIT DROP;
12
13    -- simulo inserimento della nuova propedeuticità
14    INSERT INTO propedeuticità_visitate
15    VALUES (NEW.insegnamento1, NEW.insegnamento2, NEW.corso_di_laurea);
16
17    -- inserisco le propedeuticità presenti
18    INSERT INTO propedeuticità_visitate
19    SELECT insegnamento1, insegnamento2, corso_di_laurea
20    FROM propedeuticità WHERE corso_di_laurea = NEW.corso_di_laurea;
21
22    SELECT count(*) INTO size
23    FROM propedeuticità_visitate;
24    LOOP
25        IF counter = size THEN
26            EXIT;
27        END IF;
28
29        FOR riga IN
30        SELECT pv1.ins1, pv2.ins2, pv1.cdl
31        FROM propedeuticità_visitate pv1 INNER JOIN
32        propedeuticità_visitate pv2
```

```

32         ON pv1.ins1 <> pv2.ins1 AND pv1.ins2 <> pv2.ins2
33     WHERE pv2.ins1 = pv1.ins2
34     LOOP
35         IF riga.ins1 = riga.ins2 THEN
36             RAISE EXCEPTION 'Propedeuticit   circolare non
37                 consentita';
38         END IF;
39         INSERT INTO propedeuticit  _visitate
40         VALUES (riga.ins1, riga.ins2, NEW.corso_di_laurea);
41     END LOOP;
42     counter := counter + 1;
43 END LOOP;
44
45 -- VERIFICA FINALE
46 PERFORM *
47 FROM propedeuticit  _visitate
48 WHERE cdl = NEW.corso_di_laurea
49     AND ins1 = NEW.insegnamento2
50     AND ins2 = NEW.insegnamento1;
51 IF FOUND THEN
52     RAISE EXCEPTION 'Propedeuticit   circolare non consentita';
53 ELSE
54     RETURN NEW;
55 END IF;
56 END;
57 $$ LANGUAGE plpgsql;

```

### 3.2.2 Carriera completa con tutti gli insegnamenti

Pu  essere utile produrre la carriera di uno studente che contenga non solamente gli insegnamenti di cui si   sostenuto almeno una volta l'esame, ma pi  in generale, un elenco che contenga tutti gli insegnamenti di cui lo studente debba sostenere l'esame per conseguire il titolo di laurea: in sintesi, sar  una **carriera\_completa** con l'inserimento ulteriore degli esami non sostenuti con la dicitura "non sostenuto". Di seguito il codice sorgente che produce quanto appena spiegato:

```

                                carriera_completa_tutti()
1 CREATE OR REPLACE FUNCTION carriera_completa_tutti(TARGET varchar) RETURNS TABLE
2 (
3     studente varchar,
4     nomstu varchar,
5     cogstu varchar,
6     cdl varchar,
7     matr integer,
8     codins varchar,
9     nomins varchar,
10    nomdoc varchar,
11    cogdoc varchar,
12    voto varchar,
13    data varchar
14 ) AS $$
15 BEGIN
16     RETURN QUERY
17     SELECT ic.studente,
18            u2.nome nomstu,
19            u2.cognome cogstu,

```

```

19         cdl.nome cdl,
20         s.matricola matr,
21         ic.insegnamento,
22         i.nome,
23         u.nome nomedoc,
24         u.cognome cogndoc,
25         CASE
26             WHEN c.valutazione IS NULL THEN 'non sostenuto'
27             ELSE c.valutazione::VARCHAR
28         END,
29         CASE
30             WHEN c.data IS NULL THEN 'non sostenuto'
31             ELSE c.data::VARCHAR
32         END
33     FROM insegnamenti_per_carriera ic
34         LEFT JOIN carriera c ON ic.insegnamento = c.insegnamento
35         INNER JOIN docente_responsabile d ON d.insegnamento =
36             ic.insegnamento
37         INNER JOIN utente u ON d.docente = u.email
38         INNER JOIN utente u2 ON ic.studente = u2.email
39         INNER JOIN studente s ON ic.studente = s.utente
40         INNER JOIN corso_di_laurea cdl ON s.corso_di_laurea = cdl.codice
41         INNER JOIN insegnamento i ON ic.insegnamento = i.codice
42     WHERE ic.studente = TARGET and c.studente = TARGET;
43 END;
44 $$ LANGUAGE plpgsql;

```

### 3.2.3 I tempi della propedeuticità

È importante che se un insegnamento A è propedeutico ad un insegnamento B per un corso di laurea C, l'anno in cui viene erogato A in C non sia successivo all'anno in cui viene erogato B sempre in C. Per controllare questa condizione, c'è il seguente trigger

#### tempi\_di\_propedeuticità

```

1 CREATE OR REPLACE TRIGGER tempi_di_propedeuticità
2 BEFORE INSERT OR UPDATE ON propedeuticità
3 FOR EACH ROW
4 EXECUTE FUNCTION propedeuticità_a_prima_b_dopo();

```

che richiama l'esecuzione della seguente funzione:

#### propedeuticità\_a\_prima\_b\_dopo()

```

1 CREATE OR REPLACE FUNCTION propedeuticità_a_prima_b_dopo() RETURNS TRIGGER AS $$
2 DECLARE
3     anno_ins1 INTEGER;
4     anno_ins2 INTEGER;
5 BEGIN
6     SELECT anno INTO anno_ins1
7     FROM insegnamento_parte_di_cdl
8     WHERE insegnamento = NEW.insegnamento1 AND corso_di_laurea =
9         NEW.corso_di_laurea;
10
11     SELECT anno INTO anno_ins2
12     FROM insegnamento_parte_di_cdl

```



```

12 WHERE insegnamento = NEW.insegnamento2 AND corso_di_laurea =
    NEW.corso_di_laurea;
13
14 IF (anno_ins1 > anno_ins2) THEN
15     RAISE NOTICE 'ATTENZIONE: un insegnamento A non puo'' essere propedeutico
        ad un insegnamento B in un Corso di Laurea C se A erogato in un anno
        successivo a B in C';
16     RAISE LOG 'ATTENZIONE: un insegnamento A non puo'' essere propedeutico ad
        un insegnamento B in un Corso di Laurea C se A erogato in un anno
        successivo a B in C';
17     PERFORM pg_notify('notifica', 'ATTENZIONE: un insegnamento A non puo''
        essere propedeutico ad un insegnamento B in un Corso di Laurea C se A
        erogato in un anno successivo a B in C');
18     RETURN NULL;
19 ELSE
20     RETURN NEW;
21 END IF;
22 END;
23 $$ LANGUAGE plpgsql;

```

### 3.2.4 La data dell'esame

Per mantenere aderenza alla realtà, non è possibile che il docente calendarizzi un esame per una data anteriore a quella attuale: per controllare questa condizione, c'è il seguente trigger

data\_non\_anteriore

```

1 CREATE OR REPLACE TRIGGER data_non_anteriore
2 BEFORE INSERT OR UPDATE ON calendario_esami
3 FOR EACH ROW
4 EXECUTE FUNCTION data_esame_non_retro();

```

che richiama l'esecuzione della seguente funzione:

data\_esame\_non\_retro()

```

1 CREATE OR REPLACE FUNCTION data_esame_non_retro() RETURNS TRIGGER AS $$
2 DECLARE
3     adesso TIMESTAMP := CURRENT_TIMESTAMP;
4     esame TIMESTAMP;
5 BEGIN
6     esame := NEW.data + NEW.ora;
7
8     IF (esame >= adesso) THEN
9         RETURN NEW;
10    ELSE
11        RAISE NOTICE 'ATTENZIONE: non puoi inserire un esame in una data e/o ora
            gia'' passata';
12        RAISE LOG 'ATTENZIONE: non puoi inserire un esame in una data e/o ora gia''
            passata';
13        PERFORM pg_notify('notifica', 'ATTENZIONE: non puoi inserire un esame in
            una data e/o ora gia'' passata');
14        RETURN NULL;
15    END IF;
16 END;
17 $$ LANGUAGE plpgsql;

```

### 3.2.5 Disiscriversi da un esame già verbalizzato

Non è consentito a uno studente di effettuare la cancellazione dall'iscrizione di un esame qualora l'esito di tale esame sia già stato verbalizzato dal docente. Non solo la GUI impedisce tale azione mostrando il pulsante inibito con la scritta "ESITO GIÀ VERBALIZZATO", ma è presente anche il trigger

no\_disiscr\_se\_verbalizzato

```
1 CREATE OR REPLACE TRIGGER no_disiscr_se_verbalizzato
2   BEFORE DELETE ON iscrizione
3   FOR EACH ROW
4   EXECUTE FUNCTION verifica_se_verbalizzato();
```

che richiama l'esecuzione della seguente funzione:

verifica\_se\_verbalizzato()

```
1 CREATE OR REPLACE FUNCTION verifica_se_verbalizzato() RETURNS TRIGGER AS $$
2
3 BEGIN
4   PERFORM *
5   FROM carriera c INNER JOIN calendario_esami ce ON ce.insegnamento =
6     c.insegnamento AND ce.data = c.data
7   WHERE c.studente = OLD.studente AND ce.id= OLD.esame;
8   IF FOUND THEN
9     RAISE NOTICE 'ATTENZIONE: non puoi cancellare l''iscrizione di un esame
10      gia'' verbalizzato';
11     RAISE LOG 'ATTENZIONE: non puoi cancellare l''iscrizione di un esame gia''
12      verbalizzato';
13     PERFORM pg_notify('notifica', 'ATTENZIONE: non puoi cancellare
14      l''iscrizione di un esame gia'' verbalizzato');
15     RETURN NULL;
16   ELSE
17     RETURN OLD;
18   END IF;
19 END;
20 $$ LANGUAGE plpgsql;
```

## 4 Prove di funzionamento

Assumendo il fatto che ciò che è consentito a basso livello (terminale e comandi SQL-DML) è ciò che viene anche consentito ad alto livello (GUI realizzata tramite HTML con interazioni grazie a PHP e Javascript), le prove di funzionamento verranno esibite mostrando soprattutto schermate del terminale in cui si prova il funzionamento di quanto descritto finora, per la corretta esecuzione delle operazioni previste. Più precisamente verranno mostrate:

- schermate dell'**interfaccia grafica** realizzata tramite HTML nel momento in cui vi è un inserimento di dati
- schermate del **terminale** nel momento in cui si vuole provare la consistenza della base di dati dopo alcune operazioni. Verranno esaminate le operazioni più critiche che potrebbero compromettere la correttezza della rappresentazione della situazione reale nell'astrazione della base di dati.

Si cercherà, lungo l'esposizione di seguire un filo cronologico che possa rispecchiare l'effettivo utilizzo dell'utente che per la prima volta utilizza il presente software. Inoltre verrà mostrato il funzionamento di PiGEU nell'ottica dell'adesione alle specifiche, e anche altre prove di funzionamento che si ritiene opportuno mostrare.

### 4.1 L'inserimento di un utente

La prima operazione da compiere per poter usufruire delle potenzialità di PiGEU è popolare la base di dati con utenti, ricordando che tra i tre diversi tipi di utenza, *Segreteria*, *Docente*, *Studente*, solo la prima tipologia è autorizzata alla gestione delle utenze, nelle operazioni di inserimento, modifica e rimozione (consideriamo lo spostamento dello studente in *storico* come un particolare tipo di rimozione)

#### 4.1.1 Studente

Proviamo ora ad inserire un nuovo **studente** e verifichiamo come si comporta la base di dati dopo il suo inserimento come nuovo utente.

Lo studente che prendiamo come modello è `demo_studente@studenti.unimi.it`. Come mostrato nella schermata sottostante, tale studente in un primo momento non è presente nella base di dati:

```
font@tuft:~$
pigeu=# select * from utente where email = 'demo_studente@studenti.unimi.it';
email | nome | cognome | indirizzo | citta | codicefiscale | emailpersonale
-----+-----+-----+-----+-----+-----+-----
(0 rows)

pigeu=# select * from studente where utente = 'demo_studente@studenti.unimi.it';
utente | matricola | corso_di_laurea
-----+-----+-----
(0 rows)

pigeu=# select * from carriera where studente = 'demo_studente@studenti.unimi.it';
studente | insegnamento | valutazione | data
-----+-----+-----+-----
(0 rows)

pigeu=# select * from insegnamenti_per_carriera where studente = 'demo_studente@studenti.unimi.it';
studente | insegnamento | timestamp
-----+-----+-----
(0 rows)

pigeu=#
```

Ora, dopo aver effettuato l'accesso come utente di segreteria, inseriamo il nuovo utente avente tipologia *studente*, come iscritto al corso di laurea in *informatica*

Nome: Demo

Cognome: Studente

CODICE FISCALE: DDMNSTD

Indirizzo: Via dimostrazione 0

Città: Proof

Selezione un'utenza: Studente

Corso di Laurea a cui appartiene questo studente: Informatica

Indirizzo email istituzionale e username di Istituto: demo\_studente @studenti.unimi.it

Indirizzo email personale: è l'account di recupero e la prima password di default dell'utente: demo@demo.demo

[INSERISCI UTENTE](#)

[RITORNA ALLA HOMEPAGE](#)

Ed ecco mostrata la situazione all'interno della base di dati successivamente l'inserimento:

```

pi@pi# select * from utente where email = 'demo_studente@studenti.unimi.it';
-----
email | nome | cognome | indirizzo | città | codicefiscale | emailpersonale
-----
demo_studente@studenti.unimi.it | Demo | Studente | Via dimostrazione 0 | Proof | DMWSTD | demo@demo.deno
(1 row)

pi@pi# select * from studente where utente = 'demo_studente@studenti.unimi.it';
-----
utente | matricola | corso_di_laurea
-----
demo_studente@studenti.unimi.it | 107 | FIX
(1 row)

pi@pi# select * from insegnamenti_per_carriera where studente = 'demo_studente@studenti.unimi.it';
-----
studente | insegnamento | timestamp
-----
demo_studente@studenti.unimi.it | 51.23.1 | 2023-08-14 14:09:55.085238
demo_studente@studenti.unimi.it | 55.23.1 | 2023-08-14 14:09:55.085238
demo_studente@studenti.unimi.it | 59.23.1 | 2023-08-14 14:09:55.085238
demo_studente@studenti.unimi.it | 77.23.1 | 2023-08-14 14:09:55.085238
demo_studente@studenti.unimi.it | 97.23.1 | 2023-08-14 14:09:55.085238
demo_studente@studenti.unimi.it | 116.23.1 | 2023-08-14 14:09:55.085238
demo_studente@studenti.unimi.it | 56.23.1 | 2023-08-14 14:09:55.085238
demo_studente@studenti.unimi.it | 115.23.1 | 2023-08-14 14:09:55.085238
demo_studente@studenti.unimi.it | 88.23.1 | 2023-08-14 14:09:55.085238
demo_studente@studenti.unimi.it | 52.23.1 | 2023-08-14 14:09:55.085238
demo_studente@studenti.unimi.it | 125.23.1 | 2023-08-14 14:09:55.085238
demo_studente@studenti.unimi.it | 98.23.1 | 2023-08-14 14:09:55.085238
demo_studente@studenti.unimi.it | TEST | 2023-08-14 14:09:55.085238
(13 rows)

pi@pi# select * from carriera where studente = 'demo_studente@studenti.unimi.it';
-----
studente | insegnamento | valutazione | data
-----
(0 rows)

pi@pi#

```

La rappresentazione dei dati è corretta: ora troviamo un nuovo utente con `nome`, `cognome`, `email` e gli altri dati correttamente settati nella tabella `utente`. Siccome il tipo di utente è uno *studente*, esso è presente anche nella tabella `studente` dove viene attribuita anche una `matricola` e vi è l'indicazione di appartenenza a un `corso_di_laurea`. Allo studente `Demo` è inoltre stato assegnato un elenco di insegnamenti di cui conseguire profitto per ottenere il titolo di laurea desiderato: ciascun `insegnamento` della tabella `insegnamenti_per_carriera` è preso dalla tabella `insegnamento parte di cdl`

#### 4.1.2 Docente

La seconda macro-entità coinvolta in PiGEU è il *docente*: in questa sede ci avvarremo di `demo_docente@unimi.it`. Anche in questo caso l'utente non è presente nella base di dati, e all'atto dell'inserimento decidiamo che sarà un docente *associato di informatica*

```

pi@pi# select * from utente where email = 'demo_docente@unimi.it';
-----
email | nome | cognome | indirizzo | città | codicefiscale | emailpersonale
-----
demo_docente@unimi.it | Demo | Docente | Via dimostrazione 0 | Proof | DMWDCT | demo@demo.deno
(1 row)

pi@pi# select * from docente where utente = 'demo_docente@unimi.it';
-----
utente | tipo
-----
demo_docente@unimi.it | associato
(1 row)

pi@pi# select * from docente_responsabile where docente = 'demo_docente@unimi.it';
-----
docente | insegnamento
-----
(0 rows)

pi@pi#

```

Vediamo che vi è consistenza nella base di dati: ora abbiamo un nuovo utente e anche un nuovo docente; l'ultima interrogazione alla base di dati è fatta appositamente per mostrare che, allo stato attuale, `demo_docente@unimi.it` non è responsabile di nessun *insegnamento*: tale riga della tabella `docente_responsabile` verrà aggiunta all'atto della creazione di un nuovo *insegnamento*,

nella omonima tabella: all'aggiunta di un nuovo inserimento verrà specificato anche il docente responsabile.

## 4.2 L'inserimento di un nuovo insegnamento

Per continuare a popolare la base di dati è opportuno inserire anche degli insegnamenti: qui inseriremo *InsProva* all'interno del corso di laurea di *Dimostrazione*, notando che al momento non è ancora presente l'insegnamento nel corso di laurea, non essendo presente nemmeno l'insegnamento stesso:

```
Font@tuf:~$ psql -h localhost -U postgres -d postgres
psql=# select * from corso_di_laurea where nome = 'Dimostrazione';
codice | nome | tipo
-----+-----+-----
DIM | Dimostrazione | triennale
(1 row)

psql=# select * from insegnamento_parte_di_cdl where insegnamento = 'DIM';
insegnamento | corso_di_laurea | anno
-----+-----+-----
(0 rows)

psql=# select * from insegnamento where nome = 'InsProva';
codice | nome | descrizione | cfu
-----+-----+-----+-----
(0 rows)

psql=#
```

allora procediamo come segue, notando che all'atto dell'inserimento di un *insegnamento* si può specificare anche una sua *propedeuticità*

ottenendo così lo stato attuale:

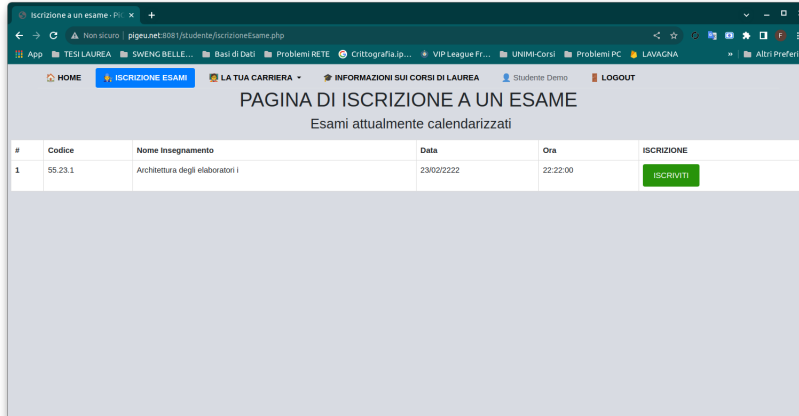
```
Font@tuf:~$ psql -h localhost -U postgres -d postgres
psql=# select * from insegnamento_parte_di_cdl where corso_di_laurea = 'DIM';
insegnamento | corso_di_laurea | anno
-----+-----+-----
IP | DIM | 1
(1 row)

psql=# select * from insegnamento where codice = 'IP';
codice | nome | descrizione | cfu
-----+-----+-----+-----
IP | InsProva | Questo è un insegnamento per mostrare la funzionalità della base di dati | 12
(1 row)

psql=#
```

### 4.3 L'iscrizione a un esame

Una delle richieste fatte nelle specifiche indica che lo *studente* deve potersi iscrivere ad esami del **solo** proprio corso di laurea. Per costruzione dell'interfaccia grafica lo studente può visualizzare il bottone di iscrizione solamente per esami di insegnamenti del proprio corso di laurea:



#### 4.3.1 Iscrizione all'esame di un insegnamento al di fuori del corso di laurea

Ora possiamo provare a "forzare" l'iscrizione dal momento che notiamo che nella base di dati non sono presenti in calendario esami di altri insegnamenti,

```
Font@tuf:~$ p1geu=# select * from calendario_esami;
+-----+-----+-----+-----+
| insegnamento | data | ora | id |
+-----+-----+-----+-----+
| 55.23.1       | 2222-02-23 | 22:22:00 | 51 |
+-----+-----+-----+-----+
(1 row)

Font@tuf:~$ p1geu=#
```

ne possiamo fare aggiungere alcuni al nostro `demo_docente@unimi.it` ottenendo così la seguente situazione:

```
Font@tuf:~$ p1geu=# select * from calendario_esami;
+-----+-----+-----+-----+
| insegnamento | data | ora | id |
+-----+-----+-----+-----+
| 55.23.1       | 2222-02-23 | 22:22:00 | 51 |
+-----+-----+-----+-----+
(1 row)

Font@tuf:~$ p1geu=# select * from calendario_esami;
+-----+-----+-----+-----+
| insegnamento | data | ora | id |
+-----+-----+-----+-----+
| 55.23.1       | 2222-02-23 | 22:22:00 | 51 |
| IP           | 2023-08-15 | 15:50:00 | 52 |
| IP           | 2023-08-25 | 17:50:00 | 53 |
+-----+-----+-----+-----+
(3 rows)

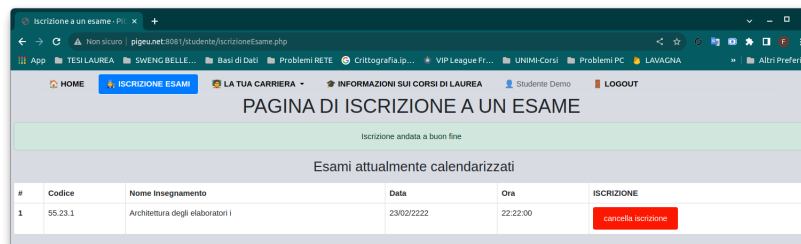
Font@tuf:~$ p1geu=#
```

adesso abbiamo altri due esami calendarizzati non appartenenti al corso di laurea in *informatica* in cui è iscritto `demo_studente@studenti.unimi.it` e siccome il nostro `demo_studente@studenti.unimi.it` non "vede" gli esami calendarizzati degli insegnamenti al di fuori del suo corso di laurea



dobbiamo tentare l'iscrizione da terminale così da vedere se veramente il comportamento esibito dalla base di dati è quello atteso, eseguendolo nei seguenti step:

- iscrizione corretta ad un esame di un insegnamento che fa parte del corso di laurea



ottenendo così la situazione seguente

```
font@tuf:~$ p1geu=# select * from calendario_esami;
+-----+
| insegnamento | data      | ora      | id |
+-----+
| 55.23.1       | 2222-02-23 | 22:22:00 | 51 |
| IP           | 2023-08-15 | 15:58:00 | 52 |
| IP           | 2023-08-25 | 17:50:00 | 53 |
+-----+
(3 rows)

p1geu=# select * from iscrizione;
+-----+
| studente | esame |
+-----+
(0 rows)

p1geu=# select * from iscrizione;
+-----+
| studente      | esame |
+-----+
| demo_studente@studenti.unimi.it | 51    |
+-----+
(1 row)

p1geu=#
```



- tentativo di iscrizione ad un esame non consentito

```

font@tuf:~$ psql -h pinguino -U pinguino -d pinguino
psql=# select * from iscrizione;
 studente | esame
-----+-----
 demo_studente@studenti.unimi.it | 51
(1 row)

psql=# INSERT INTO iscrizione (studente, esame) VALUES ('demo_studente@studenti.unimi.it', 53);
NOTICE: ATTENZIONE: non è consentita l'iscrizione ad un esame che non appartiene al corso di laurea di uno studente
INSERT 0 0
psql=#

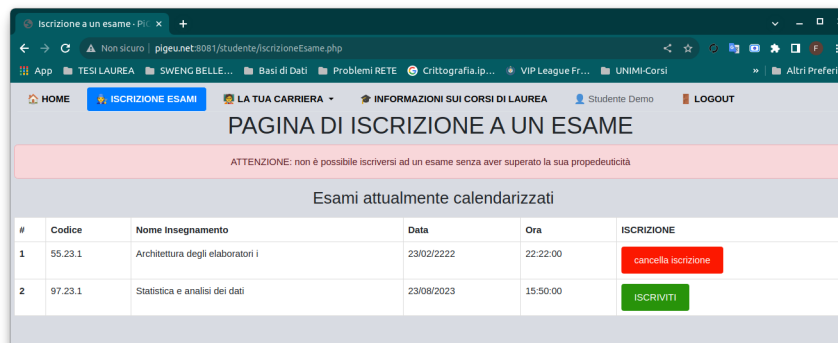
```

con il conseguente sollevamento dell'informazione da parte del trigger che comunica il messaggio personalizzato impostato nella dichiarazione dello stesso.

### 4.3.2 Iscrizione senza rispettare la propedeuticità

L'iscrizione ad un esame di cui non si è rispettata la propedeuticità viene impedita sia da interfaccia grafica, sia da terminale a causa di un trigger che controlla il rispetto delle propedeuticità. Prendiamo come esempio il corso di *informatica* in cui l'insegnamento di *Matematica del Continuo* è propedeutico all'insegnamento di *Statistica e analisi dei dati* Nella fattispecie:

- Nell'interfaccia grafica, qualora si clicchi sul pulsante "iscriviti", viene mostrato un messaggio di errore sollevato dal trigger e catturato dallo script PHP che riflette tale messaggio a schermo:



- Da terminale invece, viene mostrato il messaggio del trigger, lo stesso mostrato prima in interfaccia grafica:

```

font@tuf:~$ psql -h pinguino -U pinguino -d pinguino
psql=# select * from calendario_esami;
 insegnamento | data | ora | id
-----+-----+----+---
 55.23.1       | 2222-02-23 | 22:22:00 | 51
 IP            | 2023-08-15 | 15:50:00 | 52
 IP            | 2023-08-25 | 17:50:00 | 53
 97.23.1       | 2023-08-23 | 15:50:00 | 54
(4 rows)

psql=# INSERT INTO iscrizione (studente, esame) VALUES ('demo_studente@studenti.unimi.it', 54);
NOTICE: ATTENZIONE: non è possibile iscriversi ad un esame senza aver superato la sua propedeuticità
INSERT 0 0
psql=#

```

## 4.4 Calendarizzare un esame

Abbiamo già visto che con PiGEU si possono calendarizzare esami, ma nelle specifiche viene chiesto che venga inibita la possibilità di calendarizzare nella stessa data, due esami dello stesso anno di un corso di laurea. Ecco un estratto della situazione attuale per quanto riguarda gli insegnamenti:

```
font@tuf: ~  
pigeu=# SELECT ic.corso_di_laurea laurea, i.codice, i.nome, ic.anno FROM insegnamento i  
INNER JOIN insegnamento_parte_di_cdl ic ON ic.insegnamento = i.codice ORDER BY laurea, anno LIMIT 20;  
laurea | codice | nome | anno  
-----+-----+-----+-----  
DIM | IP | InsProva | 1  
F1X | 116.23.1 | Linguaggi Formali e Automi | 1  
F1X | 55.23.1 | Architettura degli elaboratori i | 1  
F1X | 59.23.1 | Matematica del Continuo | 1  
F1X | 77.23.1 | Architettura degli elaboratori ii | 1  
F1X | 88.23.1 | Matematica del Discreto | 1  
F1X | 115.23.1 | Logica matematica | 1  
F1X | 56.23.1 | Programmazione | 1  
F1X | 51.23.1 | Basi di dati | 2  
F1X | 52.23.1 | Algoritmi e strutture dati | 2  
F1X | 98.23.1 | Sistemi operativi | 2  
F1X | 97.23.1 | Statistica e analisi dei dati | 2  
F1X | 125.23.1 | Programmazione ii | 2  
F1X | TEST | Insegnamento | 3  
F1XM | 77.23.1 | Architettura degli elaboratori ii | 1  
F1XM | 52.23.1 | Algoritmi e strutture dati | 1  
F1XM | TEST | Insegnamento | 2  
G1 | 97.23.1 | Statistica e analisi dei dati | 1  
G1 | 59.23.1 | Matematica del Continuo | 1  
G1 | 77.23.1 | Architettura degli elaboratori ii | 1  
(20 rows)  
pigeu=#
```

E per quanto riguarda il calendario esami:

```
font@tuf: ~  
pigeu=# select * from calendario_esami;  
insegnamento | data | ora | id  
-----+-----+-----+-----  
55.23.1 | 2222-02-23 | 22:22:00 | 51  
IP | 2023-08-15 | 15:50:00 | 52  
IP | 2023-08-25 | 17:50:00 | 53  
97.23.1 | 2023-08-23 | 15:50:00 | 54  
(4 rows)  
pigeu=#
```

Notiamo subito che il trigger impostato per non permettere nella stessa giornata due esami di insegnamenti dello stesso anno dello stesso corso di laurea dovrebbe inibire la possibilità all'insegnamento di *Matematica del Continuo* di inserire un esame per la data del 23-02-2222 poiché in tale data c'è già programmato un esame di *Architettura degli elaboratori i*.

- Nell'interfaccia grafica, cliccando su "inserisci" l'esame non viene calendarizzato e viene mostrato un messaggio di errore, che viene catturato dallo script PHP dalla **NOTICE** del trigger

- Nel terminale viene sempre inibita la possibilità di calendarizzare:

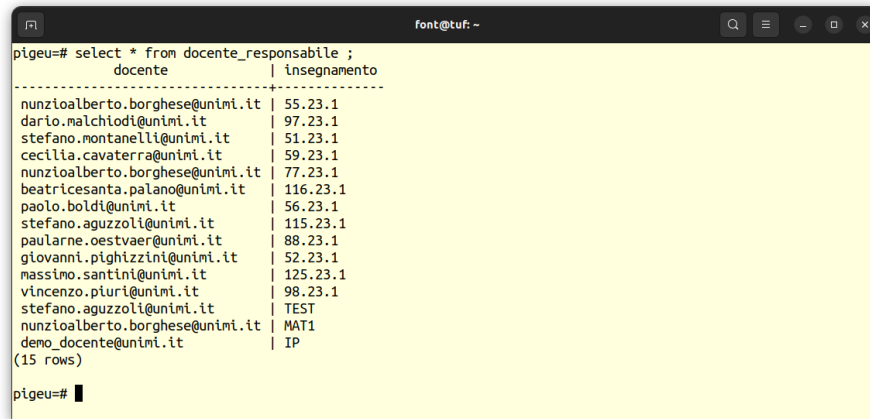
```
font@tuf: ~
pigeu=# select * from calendario_esami;
+-----+-----+-----+-----+
| insegnamento | data      | ora      | id |
+-----+-----+-----+-----+
| 55.23.1       | 2222-02-23 | 22:22:00 | 51 |
| IP            | 2023-08-15 | 15:50:00 | 52 |
| IP            | 2023-08-25 | 17:50:00 | 53 |
| 97.23.1       | 2023-08-23 | 15:50:00 | 54 |
+-----+-----+-----+-----+
(4 rows)

pigeu=# insert into calendario_esami values ('59.23.1', '2222-02-23', '15:50:00');
NOTICE:  ATTENZIONE: e' gia' presente un altro esame dello stesso anno per la data selezionata
INSERT 0 0
pigeu=#
```

## 4.5 Il docente Responsabile

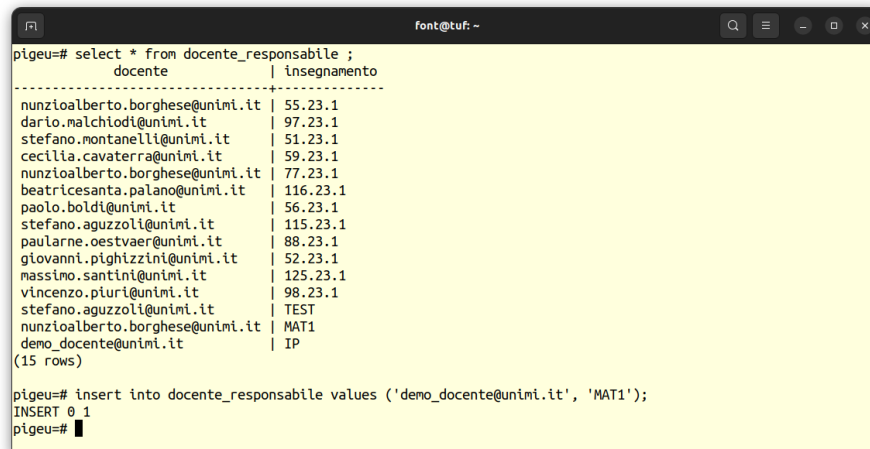
Nei requisiti viene richiesto che un docente sia responsabile di al più tre insegnamenti. Anche in questo caso proviamo a forzare la base di dati e vediamo se il trigger adibito al controllo funziona correttamente:

- Notiamo dalla seguente interrogazione che `demo_docente@unimi.it` è responsabile di un solo insegnamento, IP:



```
font@tuf: ~  
pigeu=# select * from docente_responsabile ;  
-----  
docente | insegnamento  
-----  
nunzioalberto.borghese@unimi.it | 55.23.1  
dario.malchiodi@unimi.it | 97.23.1  
stefano.montanelli@unimi.it | 51.23.1  
cecilia.cavaterra@unimi.it | 59.23.1  
nunzioalberto.borghese@unimi.it | 77.23.1  
beatricesanta.palano@unimi.it | 116.23.1  
paolo.boldi@unimi.it | 56.23.1  
stefano.aguzzoli@unimi.it | 115.23.1  
paularne.oestvaer@unimi.it | 88.23.1  
giovanni.pighizzini@unimi.it | 52.23.1  
massimo.santini@unimi.it | 125.23.1  
vincenzo.piuri@unimi.it | 98.23.1  
stefano.aguzzoli@unimi.it | TEST  
nunzioalberto.borghese@unimi.it | MAT1  
demo_docente@unimi.it | IP  
(15 rows)  
pigeu=#
```

- Proviamo allora ad inserire un ulteriore insegnamento di cui sia responsabile:



```
font@tuf: ~  
pigeu=# select * from docente_responsabile ;  
-----  
docente | insegnamento  
-----  
nunzioalberto.borghese@unimi.it | 55.23.1  
dario.malchiodi@unimi.it | 97.23.1  
stefano.montanelli@unimi.it | 51.23.1  
cecilia.cavaterra@unimi.it | 59.23.1  
nunzioalberto.borghese@unimi.it | 77.23.1  
beatricesanta.palano@unimi.it | 116.23.1  
paolo.boldi@unimi.it | 56.23.1  
stefano.aguzzoli@unimi.it | 115.23.1  
paularne.oestvaer@unimi.it | 88.23.1  
giovanni.pighizzini@unimi.it | 52.23.1  
massimo.santini@unimi.it | 125.23.1  
vincenzo.piuri@unimi.it | 98.23.1  
stefano.aguzzoli@unimi.it | TEST  
nunzioalberto.borghese@unimi.it | MAT1  
demo_docente@unimi.it | IP  
(15 rows)  
  
pigeu=# insert into docente_responsabile values ('demo_docente@unimi.it', 'MAT1');  
INSERT 0 1  
pigeu=#
```

- il comportamento atteso è infatti quello di un normalissimo inserimento che non viola ancora le condizioni dal momento che `demo_docente@unimi.it` si trova ad essere responsabile di due insegnamenti, ora:

```
font@tuf: ~  
pigeu=# select * from docente_responsabile ;  
-----  
docente | insegnamento  
-----  
nunzioalberto.borghese@unimi.it | 55.23.1  
dario.malchiodi@unimi.it | 97.23.1  
stefano.montanelli@unimi.it | 51.23.1  
cecilia.cavaterra@unimi.it | 59.23.1  
nunzioalberto.borghese@unimi.it | 77.23.1  
beatricesanta.palano@unimi.it | 116.23.1  
paolo.boldi@unimi.it | 56.23.1  
stefano.aguzzoli@unimi.it | 115.23.1  
paularne.oestvaer@unimi.it | 88.23.1  
giovanni.pighizzini@unimi.it | 52.23.1  
massimo.santini@unimi.it | 125.23.1  
vincenzo.piuri@unimi.it | 98.23.1  
stefano.aguzzoli@unimi.it | TEST  
nunzioalberto.borghese@unimi.it | MAT1  
demo_docente@unimi.it | IP  
demo_docente@unimi.it | MAT1  
(16 rows)  
pigeu=#
```

- ma cosa succede se proviamo a rendere `demo_docente@unimi.it` responsabile di altri due insegnamenti, ovvero arrivando a quattro responsabilità cercando così di violare la specifica con l'ultimo inserimento?

```
font@tuf: ~  
pigeu=# select * from docente_responsabile ;  
-----  
docente | insegnamento  
-----  
nunzioalberto.borghese@unimi.it | 55.23.1  
dario.malchiodi@unimi.it | 97.23.1  
stefano.montanelli@unimi.it | 51.23.1  
cecilia.cavaterra@unimi.it | 59.23.1  
nunzioalberto.borghese@unimi.it | 77.23.1  
beatricesanta.palano@unimi.it | 116.23.1  
paolo.boldi@unimi.it | 56.23.1  
stefano.aguzzoli@unimi.it | 115.23.1  
paularne.oestvaer@unimi.it | 88.23.1  
giovanni.pighizzini@unimi.it | 52.23.1  
massimo.santini@unimi.it | 125.23.1  
vincenzo.piuri@unimi.it | 98.23.1  
stefano.aguzzoli@unimi.it | TEST  
nunzioalberto.borghese@unimi.it | MAT1  
demo_docente@unimi.it | IP  
demo_docente@unimi.it | MAT1  
(16 rows)  
  
pigeu=# insert into docente_responsabile values ('demo_docente@unimi.it', 'TEST');  
INSERT 0 1  
pigeu=# insert into docente_responsabile values ('demo_docente@unimi.it', '97.23.1');  
NOTICE: ATTENZIONE: il docente e' gia' responsabile di tre insegnamenti  
INSERT 0 0  
pigeu=#
```

- come da comportamento desiderato, l'ultimo inserimento viene vietato, mostrando il messaggio del trigger indicante il raggiungimento della soglia massima di insegnamenti per cui un docente può essere responsabile, mantenendo così, nonostante le 4 richieste di responsabilità, la consistenza della base di dati che contiene tre responsabilità per demo\_docente@unimi.it:

```
font@tuf: ~  
(16 rows)  
pigeu=# insert into docente_responsabile values ('demo_docente@unimi.it', 'TEST');  
INSERT 0 1  
pigeu=# insert into docente_responsabile values ('demo_docente@unimi.it', '97.23.1');  
NOTICE:  ATTENZIONE: il docente e' gla' responsabile di tre insegnamenti  
INSERT 0 0  
pigeu=# select * from docente_responsabile ;  
-----  
docente | insegnamento  
-----  
nunzioalberto.borghese@unimi.it | 55.23.1  
dario.malchiodi@unimi.it | 97.23.1  
stefano.montanelli@unimi.it | 51.23.1  
cecilia.cavaterra@unimi.it | 59.23.1  
nunzioalberto.borghese@unimi.it | 77.23.1  
beatricesanta.palano@unimi.it | 116.23.1  
paolo.boldi@unimi.it | 56.23.1  
stefano.aguzzoli@unimi.it | 115.23.1  
paularne.oestvaer@unimi.it | 88.23.1  
giovanni.pighizzini@unimi.it | 52.23.1  
massimo.santini@unimi.it | 125.23.1  
vincenzo.pluri@unimi.it | 98.23.1  
stefano.aguzzoli@unimi.it | TEST  
nunzioalberto.borghese@unimi.it | MAT1  
demo_docente@unimi.it | IP  
demo_docente@unimi.it | MAT1  
demo_docente@unimi.it | TEST  
(17 rows)  
pigeu=#
```

## 5 Funzionalità *user-friendly*

### 5.1 Icona di connessione alla base di dati

Nella pagina di login, sotto al form in cui inserire le proprie credenziali, vi è un'icona indicante lo stato della connessione con la base di dati, così che l'utente che sia difficoltà all'accesso può comprendere se le cause del mancato servizio sono dovute a cause tecniche della connessione con la base di dati oppure a problematiche del dispositivo dell'utente. Le icone di stato sono due:

- un cerchio ripieno verde con la dicitura ONLINE indica la connessione corretta con la base di dati
- un cerchio ripieno rosso con la dicitura OFFLINE indica la connessione corretta con la base di dati

### 5.2 Recupero delle credenziali

Nella pagina di login, sempre sotto al form in cui inserire le proprie credenziali, c'è la dicitura "Password dimenticata" che rimanda a una pagina dove poter inserire il proprio indirizzo email di recupero, che deve corrispondere con quello impostato come email personale (non istituzionale!) all'atto dell'iscrizione. Se vi è corrispondenza tra ciò che è stato inserito e quanto è presente nella base di dati, verrà inviata una email a tale indirizzo di recupero contenente un link per reimpostare la propria password.

### 5.3 Produzione dei documenti in PDF

PiGEU dispone anche della possibilità di generare la documentazione di carriera anche in formato PDF scaricabile o stampabile. Tale funzionalità è presente sia nel profilo dello studente, che può produrre il PDF con la propria carriera (valida o completa), sia nel profilo della segreteria che può produrre il PDF con la carriera (valida o completa) di qualsiasi studente.

## 6 Sicurezza di PiGEU

### 6.1 crittografia delle credenziali

Le credenziali, fin dalla homepage di login, vengono sempre trasmesse con il metodo POST, quindi con i valori non visibili dalla barra degli indirizzi, e per quanto riguarda la password viene usata la cifratura MD5 per garantire la sicurezza. Inoltre nella base di dati, nella tabella [credenziali](#) le password sono cifrate sempre con l'algoritmo di cifratura HASH MD5.

### 6.2 autorizzazioni per determinati utenti

Di fondamentale importanza è gestire le autorizzazioni per gli utenti autenticati: se uno studente si autentica con le proprie credenziali di studente, può modificare manualmente la barra degli indirizzi oppure il codice sorgente facendo un *inspect* nel browser e inviare informazioni o richieste malevole o fraudolente come ad esempio tentare l'accesso ad una dashboard di un docente, ma procediamo per ordine.

### 6.2.1 la navbar limitata all'utente

la navbar personalizzata per ogni utenza già limita le operazioni a quel determinato dominio di applicazione che sia di tipo *studente* oppure *docente* oppure *segreteria*. Di default non dovrebbero quindi essere accessibili pagine o comandi esterni all'ambito di cui fa parte lo specifico utente

### 6.2.2 il controllo in homepage del tipo di utente

La home page, in fase di autenticazione, effettua un redirecting a seconda della tipologia di utente associata alle credenziali inserite, alla pagina idonea al tipo di utenza

### 6.2.3 controllo in testa a ogni pagina

Una parte interessante per quanto riguarda la sicurezza è proprio il controllo che viene fatto all'atto del caricamento di ogni pagina: tutte le pagine tengono in considerazione le variabili `$_SESSION` in cui sono memorizzati l'username e la password cifrata. Al caricamento di ogni pagina del tipo di utente, viene effettuata una interrogazione al database per verificare se quelle credenziali sono associate ad un tipo di utenza che sia autorizzata ad accedere alle informazioni e comandi presenti nella pagina desiderata. Ciò risolve anche i problemi di manipolazione di codice o di barra degli indirizzi descritti all'inizio di questa sezione. Se questo criterio di appartenenza di tipo non viene rispettato, PiGEU esegue un **logout forzato** eseguendo un redirecting alla home page dove inserire le credenziali.

### 6.2.4 Prevenzione attacchi DOS

La pagina di login è pensata per prevenire anche attacchi di *Denial of Service*, bloccando qualsiasi interazione con la pagina per alcuni secondi, qualora le credenziali siano errate: ciò può arginare i danni originati da *bot* programmati per fare *bruteforcing* sulle password una volta note le credenziali di un utente, per esempio. Da notare che il tempo di blocco-attesa tra un tentativo di accesso e il successivo raddoppia ogni volta che si tenta di accedere con credenziali non autorizzate

### 6.2.5 Modifica password iniziale

Nel momento in cui la segreteria inserisce una nuova utenza nella base di dati, a questa viene assegnata come password provvisoria l'indirizzo email di recupero. Quest'ultima è considerata poco affidabile in termini di sicurezza quindi al primo accesso verrà richiesto di cambiare la propria password. Non è consentito mantenere come password il proprio indirizzo email di recupero.