Documentazione Tecnica Progetto Basi Di Dati 2022-2023

Martino Francesco Leone

Agosto 2023

Contents

1 Introduzione

Di seguito, la presente documentazione tecnica procederà a illustrare, come richiesto, tutte le scelte implementative che hanno portato allo sviluppo della base di dati, insieme alle scelte di progettazione concettuale e logica della stessa, nonché gli aspetti metodologici adottati.

Verranno inoltre illustrati tutti i trigger e le funzioni implementati, e sarà fornita una definizione di come ciascun componente sia stato interpretato in relazione alla traccia d'esame[1].

Si procederà ora con la dettagliata esposizione delle suddette informazioni.

[1]Per "traccia d'esame" si fa riferimento al compito assegnato dal professore e pubblicato su Ariel, aggiornato al 25 agosto 2023; si presume che tale traccia sia nota al lettore

2 Definizione Delle Entità

Partendo dalla consegna procederemo definenendo tutte le entità che fondamentali andranno a comporre la nostra base di dati

2.1 Corso Di Laurea

2.1.1 Definizione

Un corso di laurea viene definito come un insimeme di insengnamenti in seguito al conseguimento della promozione di ognuno di essi da parte degli studenti si consegue la laurea. Un corso di laurea può essere di due tipi

- **Triennale**: la cui duratà sarà di tre anni
- **Magistrale**[2]: la cui duratà sarà di tre anni

il corso di laurea sarà struttuarato per avere validità di un determinato anno:

in modo che se vi fosse differenza di corsi tra l'erogazione dello stesso corso ma in anni diversi sarà possibile registrare questo cambiamento. discorso analogo vale per gli insegnamenti (vedi 2.2)

2.1.2 Parametri

- **nome**: nome corso
- **durata**: durata corso **2-3**
- **anno**: inteso come anno in cui gli studenti possono iscriveri al corso di laurea *(es. le matricole di informatica 2023 si iscrivernanno al corso con nome=informatica e anno=2023)*
- **descrizione**: breve descrizione del corso

2.2 Insegnamento

2.2.1 Definizione

Un insegnamento è associato ad un docente ed ad un corso di laurea.

Un insegnamento può essere associato ad un solo corso di laurea, esso avrà un docente come responsabile un annoConsigliato e il numero di cfu

2.2.2 Parametri

- **Nome Insegnamento**
- **Anno Consigliato**: anno in cui si consiglia di seguire il corso
- **cfu**: numero di cfu che uno studente ottiene alla promozione
- **Corso di Appartenenza**: laurea di appartenenza
- **responsabile**: Docente responsabile del corso

2.3 Docente

2.3.1 Definizione

Quest'entità rappresenterà ogni singolo docente con le sue generalità tra cui la password che sarà fondamentale per il login

2.3.2 Parametri

- **Email** :
- **Passoword**:
- **Nome**
- **Cognome**
- **Data Di Nascita**

[2] si assume il conseguimento di una laurea triennale prima di potersi iscrivere ad una magistrale

2.4 Studente

2.4.1 Definizione

Quest'entità rappresenterà ogni singolo Studente con le sue generalità tra cui la password che sarà fondamentale per il login.

2.4.2 Parametri

- **matricola**: codice idetificativo dello studente
- **email**: email dello studente
- **Passoword**: password dello studente
- **nome**: nome dello studente
- **cognome**: cognome dello studente
- **cfu**:numero di cfu attualmente associati allo studente defino dinamenticamente dalla funzione update_cfu (sezione 8.2.4)
- **idLaurea**: corso di laurea a cui è iscritto
- **dataN**: data di nascita

2.5 Studente Archivio

2.5.1 Definizione

Quest'entità rappresenterà ogni singolo Studente che viene rimosso dagli studenti attivi con le sue generalità tra cui la password che sarà fondamentale per il login.

2.5.2 Parametri

- **matricola**: codice identificativo dello studente
- **email**: email dello studente
- **Passoword**: password dell'utente
- **nome**: nome dello studente
- **cognome**: cognome dello studente
- **cfu**: numero di cfu coseguiti dallo studente al momento dell'archiviazione
- **idLaurea**: corso di laurea a cui è iscritto
- **dataN**: data di nascita
- **Perido di inattività**: quanto tempo è decorso dall'ultimo esame dato alla data della sua rimozione

2.6 Appello

2.6.1 Definizione

Ogni Insegnamento deve avere associati un elemento Appelo che permetta agli studenti di iscriversi ad un esame tramite il quale verranno poi valutati e potranno proseguire con la propria carriera

2.6.2 Parametri

- **data**: data dell'esame
- **luogo**: luogo dell'esame
- **corso**: corso di appartenenza

2.7 Segreteria

2.7.1 Definizione

La segreteria verrà implementata come account "manageriale" e non come un account associato ad una persona fisica, quindi tutti gli account segreteria avranno tutti quanti gli stessi permessi di creazione,rimozione e modifica dei profili Docente(2.3), Studente(2.4) ,gestione insegnamenti(2.2),corsi di laurea (2.1) ecc...

2.7.2 Parametri

Essendo account non associati a persone fisiche gli unici parametri di cui necessitano è sono:

- email
- password

3 Definizione Relazioni

Date le entità viste nel paragrafo precedente però non sono sufficenti a soddisfare la consegna. Procediamo quindi con la definizione delle relazioni,cominciamo da quelle più ovvie:

- **Insegnamento-Croso Di Laurea**: avremo una relazione 1 ad N in quanto *corsoDiAppartenenza* fungerà da chiave esterna
- **Insegnamento-Docente**: questa relazione 1 a molti permetterà di associare un insegnamento il docente ad esso responsabile

Un occhio attento perà notera che le entità fin ora mostrate non sono sufficenti per un soddisfacimento delle richieste, quindi è necessario realizzare delle altre entità che permettano la correlazioni tra le entità preesistenti.

3.1 Sostiene

3.1.1 Definizione

Si tratta di un entità che permetterà di correlare gli appelli con gli studenti consentendo quindi di iscriversi agli appelli e di soddisfare il requisito sulla gestione dei voti

3.1.2 Parametri

- **corso**: definisce il riferimento al corso
- **appello**: definisce il riferimento all'appello
- **voto**: definisce il voto che lo studente ha preso in quel determinato appello

  Si noti che in un primo momento voto sarà null al momento dell'iscrizione all'appello il suo valore cambierà solo in seguito alla correzzione del professore e all'assegnamento del voto

3.2 Propedeuticità

3.2.1 Definizione

Un altra entità che non abbiamo ancora visto è quella necessaria a soddisfare la richiesta di impleementazione della meccanica sulla propedeuticità la quale prevede che la base di dati si assicuri che prima che uno studente possa iscriveri ad un appello che egli abbia già conseguito la promozione in un altro esame propedeutico.

3.2.2 Parametri

- **insegnamento**: insegnamento normale
- **insegnamentoPropedeutico**: insegnamento che bisogna aver conseguito per poter iscriversi all'insegnamento 'insegnamento normale'

4 Gestione Archivio

Ci accingiamo ora ad introdurre le meccaniche di Gestione dell'archivio.

Definiamo per prima cosa quale sarà lo scopo dell'archivio: l'archivio sarà un insieme di entità che permetteranno al momento della rimozione di un utente di salvarne la propria intera carriera

4.1 Studente Archivio

4.1.1 Definizione

entità che permette di contenere le informazioni dello studente archiviato

4.1.2 Parametri

- **matricola**: Matricola dello studente
- **email**: Email dello studente
- **Password**: Password dello studente
- **nome**: Nome dello studente
- **cognome**: Cognome dello studente
- **cfu**: CFU dello studente
- **data nascita**: Data di nascita dello studente
- **periodo Inattività**: periodo di inattività dello studente
- **Laurea**: Laurea a cui era iscritto lo studente al momento dell'archivio

4.2 Voto Archivio

4.2.1 Definizione

una semplice entità in cui tutti i voti presenti in sostiene associati allo studente archiviato vengono trasferiti

4.2.2 Parametri

- **voto**
- **insegnamento archivio**

4.3 Insegnamento Archivio

una semplice entità in cui i campi in voti archivio vengono associati all'insegnamento a cui sono stati registrati

5 Diagramma Concettuale

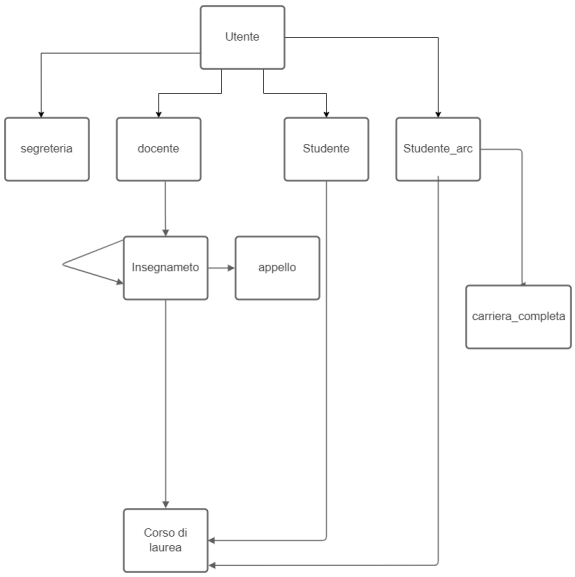

Figure 1: Prima versione del diagramma concettuale

Iniziamo a sviluppare il diagramma concettuale partendo da ciò che possiamo dedurre a una prima lettura della consegna, sapremo che avremo le seguenti entità:

- Utente:
  - Segreteria
  - Docente
  - Studente
  - Studente Archiviato
- corso di laurea
- Insegnamento

5.1 Utenti

Per prima cosa concentriamoci su utenti i quale si trovano in una gerarchia e necessità una ristrutturazione:

procederemo con una ristrutturazione verso il basso dove quindi ci assicureremo che ogni entità figlia sia indipentente

5.2 Insegnamenti

non necessità di particolari ristrutturazioni se non una specificazione che la relazione con se stessa definisce la propedeuticità e si tratta di un n-n

5.3 Corso di Laurea

non necessità ristrutturazioni

5.4 Studente

non necessità ristrutturazioni

5.5 Appello

Appello necessita di una grande ristrutturazione in quando necessità un associazione a studente e l'aggiunta di una tabella che permettà l'iscrzione agli studenti che chiameremo sostiene (3.2)

5.6 Archivio

Per quanto riguarda la gestione della tabella carriera completa segue una ristruttuazione in linea con la struttura esposta nella sezione 8.4

5.7 Aggiornamento Diagramma

applicando quindi le ristrutturazioni sopra citate possiamo illustrare il nuovo diagramma concettuale in figura 2

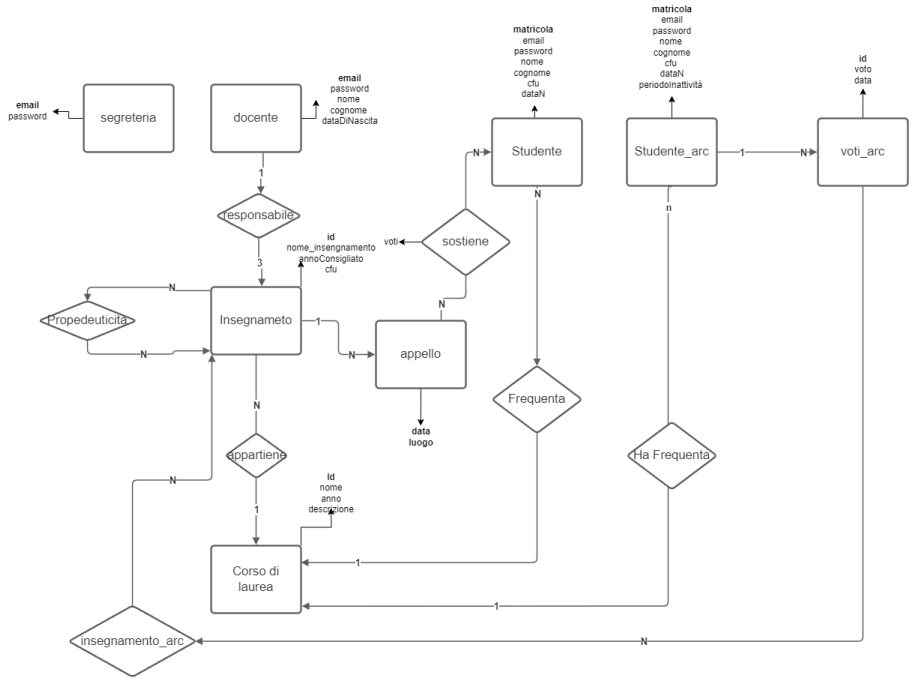

Figure 2: diagramma concettuale definitivo

Come si può notare sono state aggiunte le propedeuticità per l'insegnamento, è stata creata la relazione sostiene tra studente e appello e l'archivio è stato strutturato

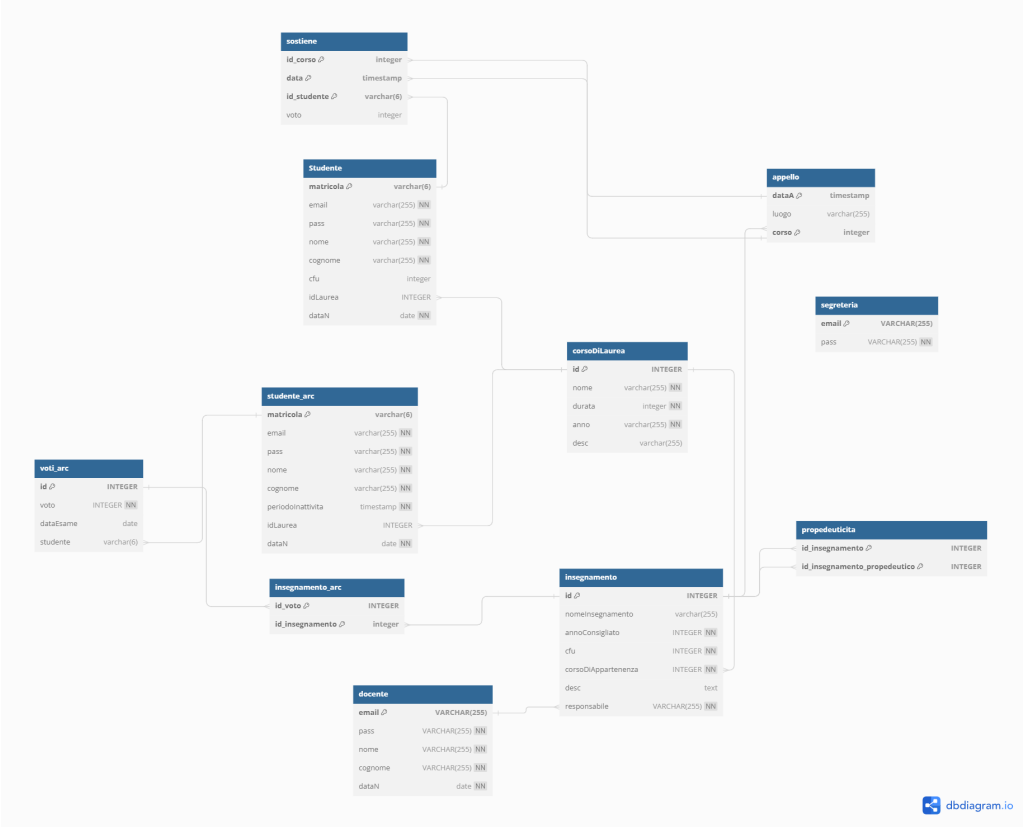

Figure 3: Diagramma Relazionale

6 Diagramma Relazionale

Procediamo ora con lo sviluppo del diagramma relazionale partendo dal concettuale, analiziamo nel dettagli tutte le scelte implementative[3] fatte in figura 3

6.1 Entità invariate

Mostriamo ora le entità che non hanno ricevuto modifice significative:

- **corsoDiLaurea**: id un integer auto increment che identifica il corso di laurea
- **Studente**: la sua struttura rimane la stessa della sezione 2.4 con la differenza che:
  - matricola: chiave primaria che identifica unviocamente lo studente
  - idLaurea: chiave esterna riferita ad un corso di laurea che permette di identificare il corso di laurea a cui è iscritto
- **docente**: email definisce unicovemente il docente con la struttura *nome.cognomeN@segreteria.com*[4]
- **Insegnamento**:
  - **id** un integer auto increment che identifica l'insegnamento
  - responsabile: chiave esterna che si riferisce all'insengnamente responsabile del corso, per motivi che illustreremo nello sezione 8.2.1 sarà fondamentale assegnare a responsabile un *CONTRAINT DEFERRABLE INITIALLY DEFERRED*
- **appello**:
  - dataA [5] : chiave primaria che rappresenta la data in cui viene sostenuto l'esame
  - corso chiave primaria di appello e chiave esterna di insegnamento
- **segreteria**: email chiave primaria (vedi 2.7 per spiegazione)

[3] ci concentreremo princpalmente sulle modifiche più radicali, molte entità infatti non verranno alterare
[4] dove N definisce un numero intero generato tra 1,100
[5] dataA è chiave primaria in modo da rispettare il vincolo del singolo esame in un giorno per corso

6.2 Moficiche

Vediamo ora le entità che hanno subito cambiamenti radicali

6.2.1 Gestione Archivio

L'archivio è gestito da una serie da alcune entità che ora vedremo e analizzeremo nel dettaglio:

- **studente_arc**: email stesso formato dello studente
- **voti_arc**: id definisce univicoamente il voto (vedi 4.2)
- **insegnamento_arc**:
  - id_voto: chiave primaria e chiave esterna su *voti_arc*
  - id_insengamento: chiave primaria e chiave esterna su *insegnamento*

  lo scopo di quest'entità emettere in relazione voti_arc e insegnamento in quanto essi hanno un rapporto n-n

6.2.2 sostiene

Quest'entità è forse quella che di più viene alterata, la sua struttua infatti diventa:

- **id_corso**: chiave primaria e chiave esterna rispetto ad appello
- **data**:chiave primaria e chiave esterna rispetto ad appello
- **id_studente**: chiave primaria e chiave esterna rispetto allo studente iscritto

7 Struttua Data Base

Definiamo quindi ora tutte le tabelle con i rispettivi campi nel dettaglio

Table 1: Tabella Docente

Campo	**Descrizione**
email	VARCHAR(255) - Chiave primaria
pass	VARCHAR(255) - Non nullo
nome	VARCHAR(255) - Non nullo
cognome	VARCHAR(255) - Non nullo
dataDiNascita	DATE

Table 2: Tabella Segreteria

Campo	**Descrizione**
email	VARCHAR(255) - Chiave primaria
pass	VARCHAR(255) - Non nullo
root	boolean - Non nullo

Table 3: Tabella CorsoDiLaurea

Campo	**Descrizione**
id	SERIAL - Chiave primaria
nome	VARCHAR(255) - Non nullo
durata	INTEGER - Non nullo
anno	VARCHAR(255) - Non nullo
desc	text - Non nullo

Table 4: Tabella Insegnamento

Campo	**Descrizione**
id	SERIAL - Chiave primaria
nomeInsegnamento	VARCHAR(255)
annoConsigliato	INTEGER - Non nullo
cfu	INTEGER
desc	TEXT
corsoDiAppartenenza	INTEGER - Chiave esterna, riferisce a "corsoDiLaurea" ("id") - Non nullo
responsabile	VARCHAR(255) - Chiave esterna, riferisce a "docente" ("email")

Table 5: Tabella Propedeuticità

Campo	**Descrizione**
id_insegnamento	INTEGER - Chiave esterna, riferisce a "insegnamento" ("id")
id_insegnamento_propedeutico	INTEGER - Chiave esterna, riferisce a "insegnamento" ("id")

Table 6: Tabella Studente

Campo	**Descrizione**
matricola	VARCHAR(6) - Chiave primaria
email	VARCHAR(255) - Unico, non nullo
pass	VARCHAR(255) - Non nullo
nome	VARCHAR(255) - Non nullo
cognome	VARCHAR(255) - Non nullo
cfu	INTEGER - Default 0
idLaurea	INTEGER - Chiave esterna, riferisce a "corsoDiLaurea" ("id")
dataN	DATE - Non nullo

Table 7: Tabella Appello

Campo	**Descrizione**
dataA	TIMESTAMP
luogo	VARCHAR(255)
corso	INTEGER - Chiave esterna, riferisce a "insegnamento" ("id")

Table 8: Tabella Sostiene

Campo	**Descrizione**
id_{corso}	INTEGER
data	TIMESTAMP
$\mathrm{id}_{studente}$	VARCHAR(6) - Chiave esterna, riferisce a "Studente" ("matricola")
voto	voto

Table 9: Tabella Studente_arc

Campo	Descrizione
matricola	VARCHAR(6) - Chiave primaria
email	VARCHAR(255) - Unico e non nullo
pass	VARCHAR(255) - Non nullo
nome	VARCHAR(255) - Non nullo
cognome	VARCHAR(255) - Non nullo
cfu	INTEGER - Predefinito a 0
periodoInattivita	INTERVAL
idLaurea	INTEGER - Chiave esterna, riferisce a "corsoDiLaurea" ("id")
dataN	DATE - Non nullo

Table 10: Tabella Voti_arc

Campo	Descrizione
id	SERIAL - Chiave primaria
voto	voto - Non nullo
dataEsame	DATE
studente	VARCHAR(6) - Chiave esterna, riferisce a "Studente_arc" ("matricola")

Table 11: Tabella Insegnamento_arc

Campo	Descrizione
id_voto	INTEGER - Chiave esterna, riferisce a "Voti_arc" ("id")
id_insegnamento	INTEGER - Chiave esterna, riferisce a "insegnamento" ("id")

8 Funzioni

Vediamo ora tutte le funzioni che sono state introdotte per permettere il matenimento corretto della base di dati:

8.1 Check:

Le seguenti funzioni (le quali saranno principalmente trigger) saranno responsabili del controllo degli inserimenti sulla base di dati, nello specifico rispetto alle entità ad esse assegnate

8.1.1 check_docente_tr

```
CREATE OR REPLACE FUNCTION check_docente()
RETURNS trigger AS $$
DECLARE
    counter INTEGER;
BEGIN
    SELECT COUNT(responsabile) INTO counter
    FROM insegnamento
    WHERE responsabile = NEW.email;

    IF counter > 0 THEN
        RETURN NEW;
    ELSE
        RAISE EXCEPTION 'docente non associato a nessun insegnamento';
    END IF;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER check_docente_insert_tr
BEFORE INSERT ON docente
FOR EACH ROW
EXECUTE FUNCTION check_docente();
```

il trigger illustrato poco sopra servirà a far si che un docente al momento prima dell'inserimento venga controllato se esso abbia o meno associato un insengamento associato al momento della crezione in caso la risposta dovesse essere negativa l'inserimento verrò annullato e verrà laciato un eccezione.

Per permettere un corretta creazione dei docenti o di altre entità "controllate" riferirsi alla sezione 8.2 in particolare alla sezione 8.2.1

8.1.2 check_insegnamento

```
CREATE OR REPLACE FUNCTION check_insegnamento(idIns integer)
RETURNS boolean AS $$
DECLARE
    ref varchar;
BEGIN
    SELECT responsabile INTO ref
    FROM insegnamento
    WHERE id = idIns;

    IF ref IS NULL THEN
        RETURN true;
    ELSE
        RETURN false;
    END IF;

END;
$$ LANGUAGE plpgsql;
```

controlla se un insegnamento abbia o meno un responsabile associato ritornando un valore boolean

8.1.3 check_esami_tr

```
CREATE OR REPLACE FUNCTION check_esami()
RETURNS trigger AS $$
DECLARE
    idLaurea INTEGER;
BEGIN
    SELECT "idLaurea" INTO idLaurea
    FROM "Studente"
    WHERE matricola = NEW.id_studente;

    IF NOT EXISTS (
        SELECT 1
        FROM insegnamento
        WHERE id = NEW.id_corso AND "corsoDiAppartenenza" = idLaurea
    ) THEN
        RAISE EXCEPTION 'insegnamento non associato al corso di laurea che si frequenta';
    END IF;

    IF NOT check_propedeuticita(NEW.id_studente, NEW.id_corso) THEN
        RAISE EXCEPTION 'propedeuticit  non rispettate';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE or replace TRIGGER check_esami_tr
BEFORE INSERT ON sostiene
FOR EACH ROW
EXECUTE FUNCTION check_esami();
```

trigger la cui funzione è quella di controllare ,prima di un inserimento all'interno della tabella *sostiene*, che lo studente soddisfi:

- le propedeuticità dell'insengamento
- l'insegnamento sia presente nel suo piano di studi

per il funzionamento di questo codice viene utilizzata la funzione *check_propedeuticita*(8.1.4)

8.1.4 check_propedeuticita

```
CREATE OR REPLACE FUNCTION check_propedeuticita(id_studenteP varchar, id_insegnamentoP int
RETURNS boolean AS $$
DECLARE
    propedeuticita_count integer;
    b integer;
BEGIN
    SELECT COUNT(*) INTO propedeuticita_count
    FROM get_propedeuticit (id_insegnamentoP);
    select COUNT(*) INTO b
    from get_propedeuticit (id_insegnamentoP) p
    inner join get_carriera_valida(id_studenteP) c on c.id_insegnamento=p.id_insegnamento;
    if (propedeuticita_count= b) then
        return true;
    else
        return false;
    end if;

END;
$$ LANGUAGE plpgsql;
```

questa funzione si occuperà di controllare se, dato una *matricola* Studente e un *id* insengnamento lo studente rispetti o meno le propedeuticità necessarie a sostenere l'esame a cui *id* fa riferimento per farlo sfrutta la funzione *get_propedeuticità* (8.3.5) controllando se gli insegnamenti propedeutici siano o meno all'interno della *get_carriera_valida* (8.3.3)

8.1.5 check_duplicate_appello

```
CREATE OR REPLACE FUNCTION check_duplicate_appello()
RETURNS TRIGGER AS $$
DECLARE
    appello_count INTEGER;
    anno_consigliato INTEGER;
    id_laurea INTEGER;
BEGIN
    SELECT "annoConsigliato", "corsoDiAppartenenza"
    INTO anno_consigliato, id_laurea
    FROM insegnamento
    WHERE id = NEW.corso;

    SELECT COUNT(*) INTO appello_count
    FROM appello
    INNER JOIN insegnamento ON appello.corso = insegnamento.id
    WHERE appello."dataA"::date = NEW."dataA"::date
    AND insegnamento."annoConsigliato" = anno_consigliato
    AND insegnamento."corsoDiAppartenenza" = id_laurea;

    IF appello_count > 0 THEN
        RAISE EXCEPTION 'Esiste gi  un appello per lo stesso giorno con
        insegnamento dell''anno consigliato';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER check_duplicate_appello_tr
BEFORE INSERT ON appello
FOR EACH ROW
EXECUTE FUNCTION check_duplicate_appello();
CREATE TRIGGER check_duplicate_appello_tr_up
BEFORE UPDATE ON appello
FOR EACH ROW
EXECUTE FUNCTION check_duplicate_appello();
```

questo trigger sulla tabella appelo si assicura che quando viene inserito o modificato un appello esso non vada in collisione con altri appelli programmati nella stessa data di un esame appartenten allo stesso corso e con lo stesso anno consigliato

8.1.6 docente_insegnamento

```
CREATE OR REPLACE FUNCTION docente_insegnamento()
RETURNS trigger AS $$
DECLARE
    counter INTEGER;
BEGIN
    SELECT count(*) INTO counter
    FROM insegnamento
    WHERE responsabile = NEW.responsabile;

    IF counter >= 3 THEN
        RAISE EXCEPTION 'Limite massimo raggiunto';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER docente_insegnamento_tr
BEFORE UPDATE or INSERT ON insegnamento
FOR EACH ROW
EXECUTE FUNCTION docente_insegnamento();
```

al momento dell'inserimento su insegnamento del responabile si assicura che il docente non abbia già 3 insegnamenti di cui è responsabile

8.1.7 check_laurea

```
CREATE OR REPLACE FUNCTION check_laurea(id_studenteP varchar)
RETURNS boolean AS $$
DECLARE
    tabella1 text[];
    tabella2 text[];
    idLaureaV INTEGER;
BEGIN
    SELECT array_agg(nome_insegnamento) FROM get_carriera_valida(id_studenteP)
    INTO tabella1;

    SELECT "idLaurea" INTO idLaureaV
    FROM "Studente"
    WHERE matricola = id_studenteP;

    SELECT array_agg(nome_insegnamento) FROM get_lista_insegnamenti(idLaureaV)
    INTO tabella2;

    IF tabella1 = tabella2 THEN
        RETURN true;
    ELSE
        RETURN false;
    END IF;
END;
$$ LANGUAGE plpgsql;
```

Questa funzione permette di controllare se uno studente soddisfi o meno le caratteristiche per riuscire richiedere la laurea ovvero se la sua carriera valida contenga tutti gli insegnamenti associati al corso di laurea che frequenta.

8.2 Create

segue un illustrazione di tutte le funzioni che posso essere utilizzate per creare campi corretti per le singole entità

8.2.1 create_docente

```
CREATE OR REPLACE FUNCTION create_docente(email varchar, pass varchar, nome varchar, cognor
RETURNS void AS $$
BEGIN
    IF check_insegnamento(id_insegnamentoP) THEN
        BEGIN
        UPDATE insegnamento
        SET responsabile = email
        WHERE id = id_insegnamentoP;
        INSERT INTO "docente" ("email", "pass", "nome", "cognome", "dataN")
        VALUES (email, pass, nome, cognome, dataDiNascita);


        COMMIT;

       END;

    ELSE
        RAISE EXCEPTION 'L''insegnamento ha gi  un docente come responsabile';
    END IF;

END;
$$ LANGUAGE plpgsql;
```

questa funzione servirà a far rispettare il trigger mostrato 8.1.1 permettendo di creare uno studente ed di assegnargli subito un responsabile. L'utilizzo di una transazione permette che il docente appena creato venga associato subito ad un insegnamento. Qui ci viene in aiuto una fattore molto importante che abbiamo definito nella tabella 4 ovvero che per rispettare il trigger *check_docente_tr* il quale controlla che al momento dell'inserimento del docente abbia già associato un insengamento. [6] quindi quello che andrà a fare la nostra funzione sarà creare una transazione che non si preoccupa degli errori fino al commiti in modo da poter agggiornare il responsabile dell'insegnamento inserendo una chiave in *responsabile* non ancora inserita nel docente.

In seguito verrà poi inserito il docente in questo modo il trigger non scatterà e potremo proseguire senza incappare in errori

8.2.2 create_insegnamento

```
1
2  CREATE OR REPLACE FUNCTION create_insegnamento(nome varchar,
3  annoConsigliato integer,corsoDiAppartenenza integer)
4  RETURNS void AS $$
5  BEGIN
6      INSERT INTO "insegnamento" ("nomeInsegnamento", "annoConsigliato",
7      "corsoDiAppartenenza", "responsabile")
8      VALUES (nome, annoConsigliato, corsoDiAppartenenza, null);
9      END;
10 $$ LANGUAGE plpgsql;
```

permettendo di creare uno studente ed di assegnargli subito un responsabile.

8.2.3 add_voto

```
1  CREATE OR REPLACE FUNCTION add_voto(votoP integer, id_studenteP varchar, id_corsoP integer
2  RETURNS void AS $$
3  BEGIN
4      UPDATE sostiene
5      SET voto = votoP
6      WHERE id_studente = id_studenteP AND id_corso = id_corsoP AND "data" = "dataP";
7      PERFORM * FROM update_cfu(id_studenteP);
8
9  END;
10 $$ LANGUAGE plpgsql;
```

questa funzione permettere di aggiungere un voto ad uno studente iscritto ad un determinato appelo

8.2.4 update_cfu

```
1  CREATE OR REPLACE FUNCTION update_cfu(id_studente varchar)
2  RETURNS void AS $$
3  DECLARE
4      cfuV integer;
5  BEGIN
6      SELECT COALESCE(sum(i.cfu), 0) INTO cfuV
7      FROM get_carriera_valida(id_studente) c
8      INNER JOIN insegnamento i ON i.id = c.id_insegnamento;
9
10     UPDATE "Studente"
11     SET cfu = cfuV
12     WHERE "Studente".matricola = id_studente;
13
14 END;
15 $$ LANGUAGE plpgsql;
```

questa funzione permette l'aggiornamento dei cfu in funzione degli esami conseguiti dallo studente

[6] questo serve a far rispettare la consegna originale del progetto che in origine prevedeva che un docente dovesse avere associato da 1 a 3 insegnamenti

8.3 Get Functions

seguono le funzioni che permettono la visualizzazione dei dati

8.3.1 get_carriera_archivio

```
CREATE OR REPLACE FUNCTION get_carriera_archivio(id_studente varchar)
RETURNS TABLE(id_insegnamento integer, nome_insegnamento varchar, voto voto, dataC date) AS
BEGIN
    RETURN QUERY
    SELECT i.id, i."nomeInsegnamento", v.voto,v."dataEsame"
    FROM insegnamento i
    JOIN insegnamento_arc i_arc ON i.id = i_arc.id_insegnamento
    JOIN voti_arc v ON i_arc.id_voto=v.id
    WHERE v.studente = id_studente;
END;
$$ LANGUAGE plpgsql;
```

ritornerà una tabella che mostrerà tutti i voti registrati per uno studente in archivio

8.3.2 get_carriera_valida_archivio

```
CREATE OR REPLACE FUNCTION get_carriera_valida_archivio(id_studente varchar)
RETURNS TABLE(id_insegnamento integer, nome_insegnamento varchar, voto voto, dataC date) AS
BEGIN
    RETURN QUERY
    SELECT i.id, i."nomeInsegnamento", v.voto, v."dataEsame"
    FROM insegnamento i
    JOIN insegnamento_arc i_arc ON i.id = i_arc.id_insegnamento
    JOIN voti_arc v ON i_arc.id_voto = v.id
    WHERE v.studente = id_studente
    AND (i.id, v."dataEsame") IN (
        SELECT i_arc.id_insegnamento, MAX(v."dataEsame")
        FROM insegnamento_arc i_arc
        JOIN voti_arc v ON i_arc.id_voto = v.id
        WHERE v.studente = id_studente
        GROUP BY i_arc.id_insegnamento
    );
END;
$$ LANGUAGE plpgsql;
```

ritornerà una tabella che mostrerà la carriera valida per uno studente in archivio

8.3.3 get_carriera_valida

```
CREATE OR REPLACE FUNCTION get_carriera_valida(id_studenteP varchar)
RETURNS TABLE (id_insegnamento integer, nome_insegnamento varchar, voto numeric,data_soste
BEGIN
    RETURN QUERY
    SELECT i.id, i."nomeInsegnamento", s.voto::numeric,s.data::date
    FROM insegnamento i
    INNER JOIN sostiene s ON i.id = s.id_corso
    WHERE s.id_studente = id_studenteP
        AND s.data = (
            SELECT MAX(data)
            FROM sostiene
            WHERE sostiene.id_corso = s.id_corso AND sostiene.id_studente = s.id_studente
        )
        AND s.voto >= 18
        AND s.voto IS NOT NULL;
    RETURN;
END;
$$ LANGUAGE plpgsql;
```

ritornerà una tabella che mostrerà la carriera valida per uno studente in archivio

8.3.4 get_carriera

```
CREATE OR REPLACE FUNCTION get_carriera(id_studenteP varchar)
RETURNS TABLE (id_insegnamento integer, nome_insegnamento varchar, voto numeric, data_soste
BEGIN
    RETURN QUERY
    SELECT i.id, i."nomeInsegnamento", s.voto::numeric,s.data::date
    FROM insegnamento i
    INNER JOIN sostiene s ON i.id = s.id_corso
    WHERE s.id_studente = id_studenteP
    AND s.voto is NOT NULL
    order by s.data;
END;
$$ LANGUAGE plpgsql;
```

ritornerà una tabella che mostrerà la carriera valida per uno studente in archivio

8.3.5 get_propedeuticità

```
CREATE OR REPLACE FUNCTION get_propedeuticit (id_insegnamentoP integer)
RETURNS TABLE (id_insegnamento integer, nome_insegnamento varchar) AS $$
BEGIN
    RETURN QUERY
    SELECT i.id, i."nomeInsegnamento"
    FROM insegnamento i
    inner join propedeuticita p on p.id_insegnamento_propedeutico=i.id
    where p.id_insegnamento=id_insegnamentoP;
    RETURN;
END;
$$ LANGUAGE plpgsql;


SELECT * FROM get_propedeuticit (5);
```

ritornerà una tabella che mostrerà la lista degli insegnamenti propedetici per un determinato insegnamento

8.3.6 get_lista_insengamenti

```
CREATE OR REPLACE FUNCTION get_lista_insegnamenti(id_laurea INTEGER)
RETURNS TABLE (id_insegnamento integer, nome_insegnamento varchar, annoConsigliato numeric)
BEGIN
    RETURN QUERY
    SELECT i.id, i."nomeInsegnamento", i."annoConsigliato"::numeric
    FROM insegnamento i
    WHERE i."corsoDiAppartenenza" = id_laurea;
END;
$$ LANGUAGE plpgsql;

SELECT * FROM get_lista_insegnamenti('1');
```

ritornerà una tabella che mostrerà la lista degli insegnamenti assegnati ad un corso di laurea

8.4 Archivio

questa sezione è dedicata a tutte le funzioni necessarie al funzionamento dell'archivio le quali sono:

- calcola_periodo_inattivita
- archivia_studente_tr

Vediamo nel dettaglio:

8.4.1 archivia_studente_tr

```
1
2  CREATE OR REPLACE FUNCTION archivia_studente_tr()
3  RETURNS TRIGGER AS
4  $$
5  DECLARE
6      periodo_inattivita INTERVAL;
7      record_voto RECORD;
8      voto_id INTEGER;
9  BEGIN
10     periodo_inattivita := calcola_periodo_inattivita(OLD.matricola);
11
12     INSERT INTO studente_arc VALUES (OLD.matricola, OLD.email,OLD.pass, OLD.nome, OLD.cogno
13
14     FOR record_voto IN SELECT * FROM get_carriera(OLD.matricola)
15     LOOP
16         IF record_voto.voto IS NOT NULL THEN
17             INSERT INTO voti_arc (id,voto,"dataEsame",studente)
18             VALUES (DEFAULT, record_voto.voto, record_voto.data_sostenimento, OLD.matricola
19             INSERT INTO insegnamento_arc(id_voto,id_insegnamento) VALUES (voto_id,record_vo
20
21         END IF;
22         DELETE FROM Sostiene WHERE id_studente=OLD.matricola;
23     END LOOP;
24
25     RETURN OLD;
26
27 END;
28 $$
29 LANGUAGE plpgsql;
30
31
32 CREATE TRIGGER before_delete_studenti
33 BEFORE DELETE ON "Studente"
34 FOR EACH ROW
35 EXECUTE FUNCTION archivia_studente_tr();
```

questo trigger si occupa di controllare quando uno studente viene rimosso dalla tabella studente e si assicura che la sua rimozione venga fatta correttamente:

1. inserisce lo studente dentro *studente_arc*
2. preleva a riga 14 tutti i voti della carriera dello *studente* che si sta eliminando e li inserisce in voti_arc assicurandosi una corrispondenza anche in insegnamento_arc
3. rimuove a riga 22 tutti campi di sostiene dove figura lo *studente*
4. infine dopo aver completato tutte queste azioni procede con la rimozione dello *studente*

8.4.2 calcola_periodo_inattivita

```
1  CREATE OR REPLACE FUNCTION calcola_periodo_inattivita(studente_id varchar)
2  RETURNS INTERVAL AS
3  $$
4  DECLARE
5      data_sostenimento TIMESTAMP;
6      is_laureato BOOLEAN;
7  BEGIN
8      SELECT check_laurea(studente_id) into is_laureato;
9      IF is_laureato THEN
10         RETURN NULL;
11     END IF;
12     -- Ottieni la data di sostenimento dell'esame pi  antico per lo studente
13     SELECT g.data_sostenimento INTO data_sostenimento
14     FROM get_carriera(studente_id) g
15     ORDER BY g.data_sostenimento ASC
```

```
16      LIMIT 1;
17
18      IF data_sostenimento IS NULL THEN
19          RETURN NULL;
20      END IF;
21
22
23      RETURN CURRENT_TIMESTAMP - data_sostenimento;
24  END;
25  $$
26  LANGUAGE plpgsql;
```

All'interno della tabella archivio troveremo colonna periodo inattività ovvero una colonna all'interno del quale definiamo quanto tempo e decorso dall'ultimo esame sostenuto all'inserimento nell'archivio

9 Esempi Esecuzione

Di seguito illustreremo il funzionamento di alcune funzioni introdotte all'interno della sezione 8

9.1 Docente

prendiamo la tabella docente (1)

fatto questo procediamo a testare la funzione 8.1.1 *check_docente_tr*, provando a lanciare la linea:

```
INSERT INTO "docente" ("email", "pass", "nome", "cognome")
VALUES ('docente@example.com', 'password', 'mario', 'rossi');
```

SQL error:

```
ERROR:  docente non associato a nessun insegnamento
CONTEXT:  PL/pgSQL function check_docente() line 11 at RAISE
```

In statement:

```
INSERT INTO "docente" ("email", "pass", "nome", "cognome")
VALUES ('docente@example.com', 'password', 'mario', 'rossi');
```

Total runtime: 5.262 ms

SQL executed.

Edit SQL

come possiamo vedere in figura 9.1 l'inserimento non è andato a buon fine in quanto il trigger è scattato e ha impedito l'inserimento.

Se invece lanciassimo la seguente linea dove creiamo un docente e gli associamo un insegnamento il cui id è 1 (dando pe assunto che l'insegnamento sia già presente nella tabella *insegnamento*

```
select * from create_docente('docen00te@example.com', 'password', 'mario',
'rossi','2010-05-01',1);
```

otteremo il seguente risultato notiamo come in questo caso tutto proceda correttamente e che il

create_docente

1 row(s)

Total runtime: 6.196 ms

SQL executed.

Edit SQL | Download

docente venga creato correttamente l'insegnamento riceva il proprio responabile,prima null, associato.

E se cercassimo di associare un docente ad un insegnamento con già un responsabile?

SQL error:

```
ERROR:  L'insegnamento ha già un docente come responsabile
CONTEXT:  PL/pgSQL function create_docente(character varying,character varying,character varying,character varying,date,integer) line 21 at RAISE
```

In statement:

```
select * from create_docente('docen05te@example.com', 'password', 'mario', 'rossi','2010-05-01',1);
```

Total runtime: 4.159 ms

SQL executed.

Edit SQL

Restando in tema della colonna responsabile è doveroso notare come se si decidesse di associare più di 3 insegnamenti ad un docente il trigger *docente_insegnamento_tr* riportado l'errore *'Limite massimo raggiunto'*

9.2 Carriera

il funzionamemto delle funzioni carriera sono identiche tra loro, cambia soltanto con che criterio sono presi i voti,per questo motivo ne mostreremo soltanto uno:

```
1  SELECT * FROM get_carriera_valida('123456');
```

id_insegnamento	nome_insegnamento	voto	data_sostenimento
2	sismologia	14	2023-08-23
2	sismologia	*NULL*	2023-08-29

2 row(s)

Total runtime: 4.545 ms

SQL executed.

Edit SQL | Download

9.3 Appello

Per la creazione di un appelle ogni volta che viene eseguito controlla che non sia in conflitto con il trigger

```
1  INSERT INTO appello VALUES ('2023-04-30','aula 200'1);
2  INSERT INTO appello VALUES ('2023-04-30','aula 202,2);
```

queste linee presuppongono chiaramente la presenza di due insegnamenti appartenenti allo stesso corso di laurea e con lo stesso anno consigliato

SQL error:

```
ERROR:  Esiste già un appello per lo stesso giorno con insegnamento dell'anno consigliato
CONTEXT:  PL/pgSQL function check_duplicate_appello() line 20 at RAISE
```

In statement:
insert into appello values ('2023-04-30 10:00:00','aula 200',1);
INSERT into appello values ('2023-04-30 10:00:00','aula 203',2);

Total runtime: 7.359 ms

SQL executed.

Edit SQL

9.4 CFU

per la gestione dei voti e dei cfu utilizziamo la funzione *add_voto* la quale peremette,se uno studente è iscritto ad un esame, di assegnare un voto nella tabella appello la quale poi prosegue aggiornando i cfu in caso di voto superiore al 18.

Ipotiziamo di avere un insegnamento con id=1 e un numero di cfu pari a 6 e di avere un appello a cui lo studente è iscritto per il 2023-08-17 10:00:00.

Questo saranno gli studenti, ci concentreremo su quello con matricola="mam": lanciando

Actions	matricola	email	pass	nome	cognome	cfu	idLaurea	dataN
Edit Delete	111111	temp	pp	pp	pp	0	2	*NULL*
Edit Delete	mam	marco.aurelio@studenti.it	1	marco	aurelio	0	1	*NULL*

2 row(s)

Expand | Insert | Refresh

quindi la seguente linea:

```
1 SELECT add_voto(30, 'mam', 1, '2023-08-17 10:00:00');
```

questa linea assegnarà il voto 30 all'appello e il risulterà nella modifica dei cfu

Actions	matricola	email	pass	nome	cognome	cfu	idLaurea	dataN
Edit Delete	111111	temp	pp	pp	pp	0	2	*NULL*
Edit Delete	mam	marco.aurelio@studenti.it	1	marco	aurelio	6	1	*NULL*

2 row(s)

Expand | Insert | Refresh