Dipartimento di ingegneria elettrica e delle tecnologie dell'informazione

U2355 - Basi di Dati

Progettazione e sviluppo di una base di dati relazionale per la gestione di corsi di formazione

Prepared by: Ariola Luigi N86003483
Instructor: Prof. Adriano Peron

Contents

List of Figures

Dizionari delle classi

1 Introduzione

1.1 Descrizione del problema

Si provvederà alla progettazione ed implementazione di una base di dati relazionale, che collabora con un sistema Java, per la memorizzazione, cancellazione e gestione di corsi di formazione per studenti. Il sistema permette ad un admin di creare uno o più corsi di formazione. Ciascun corso è caratterizzato da un nome, una descrizione, un massimo di partecipanti, un numero di presenze obbligatorie, una data di inizio e da una o più lezioni. A sua volta ogni lezione presenterà un titolo, una descrizione, una data ed ora inizio. Ogni corso potrà essere definito da una o più aree tematiche, che sono definite dagli admin del sistema. Ciascun operatore potrà iscrivere uno o più studenti ai corsi, e impostare le assenze/presenze per ogni lezione. É possibile, inoltre, controllare l'andamento di un corso attraverso una serie di statistiche, tra le quali il numero medio, massimo e minimo di studenti a lezione. Inoltre si possono verificare gli studenti che hanno raggiunto il numero di presenze obbligatorie e sono quindi idonei al superamento del corso.

2 Progettazione concettuale

2.1 Introduzione

In questa parte si inizia la progettazione del sistema ad un livello di astrazione più alto. Dopo aver analizzato il problema, si arriverà alla costruzione di un Class Diagram delle classi che descrive la struttura del sistema. Successivamente ci sarà la ristrutturazione del class diagram, ed il dizionario delle classi.

2.2 Class Diagram

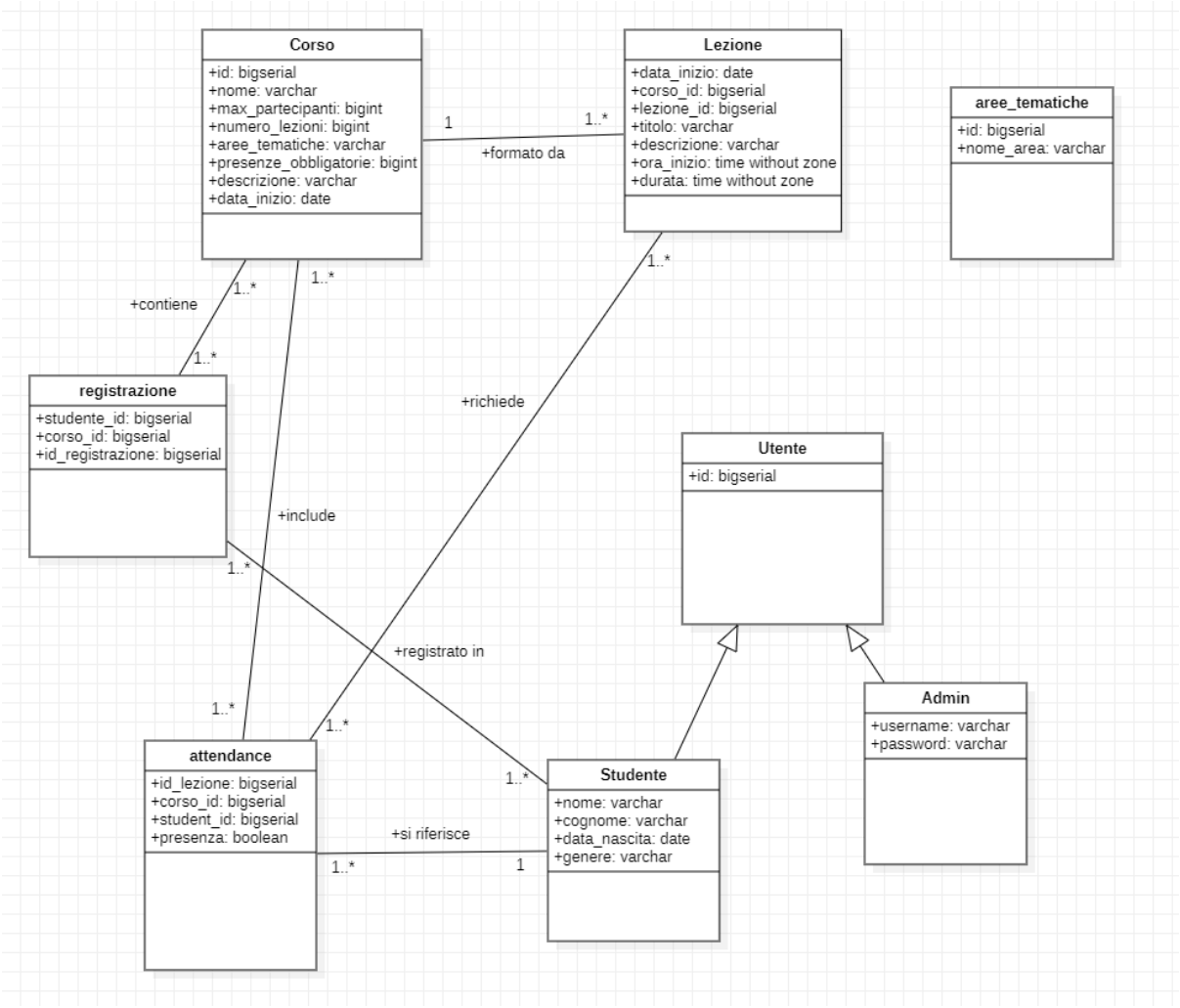

Figure 2.1: Class Diagram

2.3 Alcune precisazioni sul Class diagram

Al fine di rendere più chiara la lettura del Class diagram, si è deciso di :

- dare un solo nome alle associazioni, al posto di darne uno per direzione;
- non impostare la lunghezza degli attributi VARCHAR

2.4 Descrizione del class diagram

Osservando la figura, un primo passo che è stato fatto nella costruzione del class diagram, è la creazione di alcune classi principali ovvero : **Studente**, **Corso**, **Lezione**. Ogni corso è in relazione con le lezioni tramite l'associazione **formato da**, che identifica per ciascuno dei corsi una o più lezioni. La classe studente è una specializzazione della classe **utente**, in quanto si è pensato, inizialmente, di distinguere i due tipi di utenti che sono presenti nella base di dati. Entrambi hanno un id univoco che li distingue, ma hanno ruoli differenti. L'**admin** è la classe che contiene l'username e la password di

ogni amministratore che può effettuare l'accesso al programma, ed effettuare le diverse operazioni. Lo studente, invece, ha tutti i suoi dati principali, ovvero il nome, cognome, la data nascita ed il genere. Il passo successivo è stato di pensare a come associare ogni studente ai vari corsi. Per questo motivo si è introdotta la classe **registrazione**, che contiene le chiavi esterne di studente e corso, ed un proprio id di registrazione. É possibile così risalire ad informazioni chiave per il funzionamento del sistema, ad esempio ogni studente a quanti e quali corsi è iscritto, il numero totale di studenti iscritti ad un corso, ecc... L'associazione **registrato in** all'interno del class diagram mostra la relazione tra studente e registrazione. Inoltre, al fine di tenere traccia delle presenze per ogni lezione di un corso di tutti gli studenti iscritti, si è pensato di creare un'altra classe : **attendance**. Essa è formata da riferimenti alle chiavi primarie di lezione corso e studente, e da un'altra colonna *presenza* che indicherà con "true" o "false" la frequenza in una determinata lezione. Infine, la classe **aree_tematiche**, si occupa di memorizzare tutte le aree tematiche che verranno successivamente associate ad un corso.

2.5 Ristrutturazione del Class Diagram

Per poter procedere alla progettazione logica della base di dati, è necessario prima effettuare una ristrutturazione del class diagram. Uno schema ristrutturato, in particolare, presenta alcune restrizioni :

1. non ci sono gerarchie di generalizzazione/specializzazione
2. non ci sono attributi multipli nelle entità
3. non ci sono attributi strutturati

Una volta terminata la ristrutturazione si potrà procedere alla traduzione in schemi relazionali mediante la progettazione logica.

2.6 Chiavi primarie

L'aggiunta delle chiavi primarie ad ogni classe è stata fondamentale soprattutto per poter identificare univocamente ognuna. Un attributo di tipo id, ad esempio, rappresenta la chiave primaria delle classi Corso, studente, registrazione, lezione, admin, in modo da poterli distinguere agevolmente.

2.7 Attributi multipli

Nel class diagram mostrato non sono presenti attributi multipli.

2.8 Attributi derivati

Nel class diagram mostrato non sono presenti attributi derivati.

2.9 Attributi strutturati

Nel class diagram mostrato non sono presenti attributi strutturati.

2.10 Gerarchie di specializzazione

Nel class diagram mostrato precedentemente ci sono due specializzazioni di utente. Per eliminarle, si è scelto il metodo di accorpamento della classe utente nelle classi figlie, trasferendo gli attributi della classe padre nelle classi sottostanti. In questo caso il metodo seguente è la strada più semplice per l'eliminazione della gerarchia, in quanto utente presentava un solo attributo id.

2.11 Class diagram ristrutturato

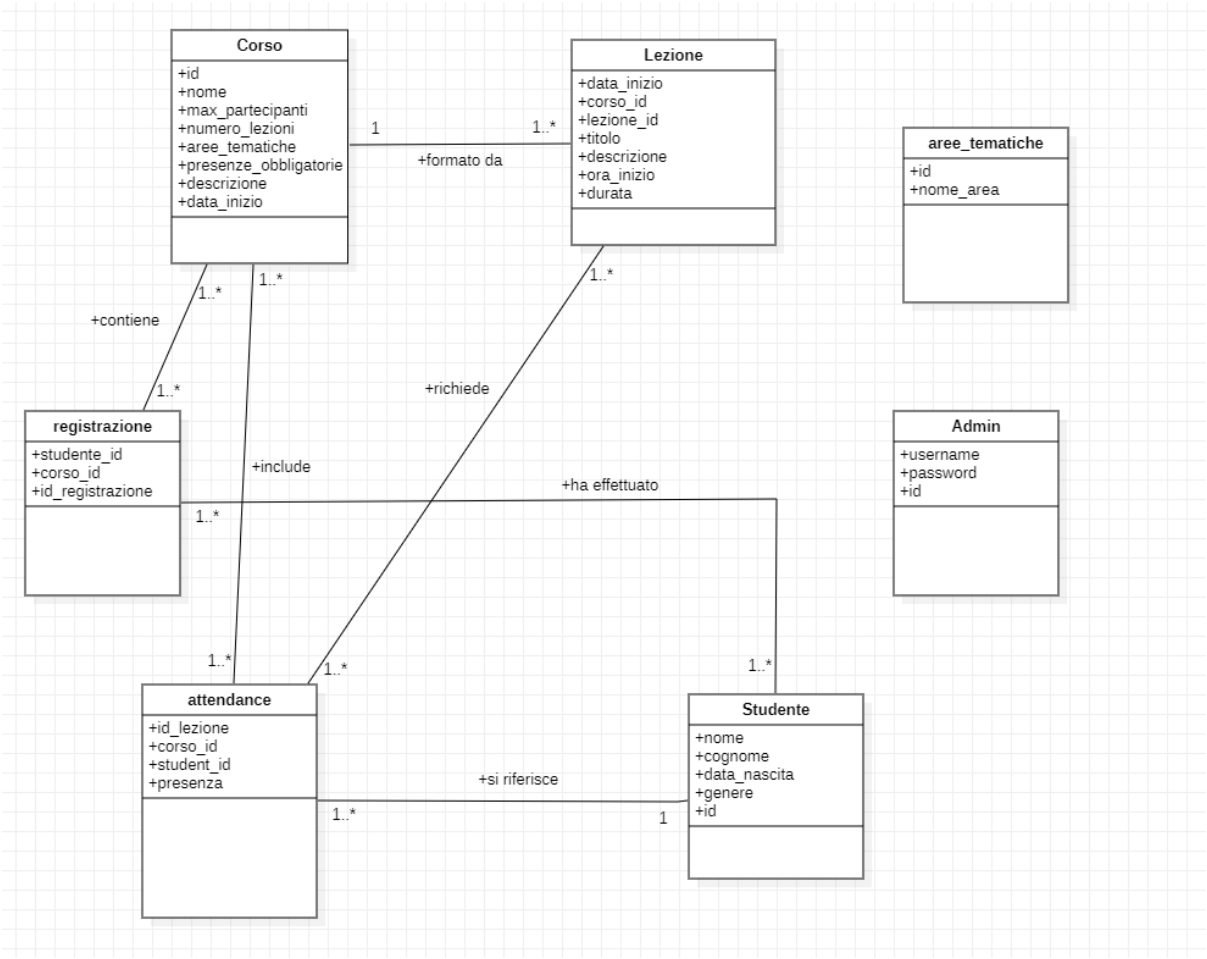

Figure 2.2: Class Diagram Ristrutturato

2.12 Dizionari

2.12.1 Dizionario delle classi

Classe	*Descrizione*	*Attributi*
Corso	Descrittore di un corso di formazione.	**id** (integer): chiave primaria. Identifica univocamente un corso. **Nome** (varchar): nome associato ad un corso. **max_partecipanti** (integer): il massimo dei partecipanti di un corso. **numero_lezioni** (integer): il numero massimo delle lezioni. **aree_tematiche** (varchar): associa una o più aree tematiche ad un corso. **presenze_obbligatorie** (integer): il minimo di presenze obbligatorie per il superamento del corso. **descrizione** (varchar): una descrizione di un corso. **data_inizio** (date): la data di inizio di un corso.
Studente	Descrittore di uno studente.	**id** (integer): chiave primaria. Identifica univocamente uno studente. **Nome** (varchar): il nome di uno studente. **Cognome** (varchar): il cognome di uno studente. **Data Nascita** (date): la data di nascita di uno studente. **Genere** (varchar): il genere di uno studente
Lezione	Descrittore di una lezione.	**data_inizio** (date): indica la data inizio di una lezione. **corso_id** (integer): attributo chiave esterna che si riferisce all'id di un corso.

Table 1: - Dizionario delle classi - cont.

Table 2: -Dizionario delle classi- prec.

Classe	*Descrizione*	*Attributi*
Lezione	Descrittore di una lezione.	**lezione_id** (integer): chiave primaria. Identifica univocamente una lezione. **titolo**: attributo che descrive il titolo assegnato ad una lezione. **corso_id** (integer): chiave esterna che referenzia corso. Identifica univocamente un corso. **descrizione** (varchar): descrizione di una lezione. **ora_inizio** (date): attributo che descrive l'ora di inizio di una lezione. **durata** (integer): attributo che identifica la durata di una lezione.
Registrazione	Descrittore di una registrazione di uno studente ad un corso.	**studente_id** (integer): attributo di chiave esterna che referenzia la chiave primaria *id* di studente. **corso_id** (integer): attributo di chiave esterna che referenzia la chiave primaria *id* di corso. **id_registrazione** (integer): attributo di chiave primaria della classe registrazione, per identificare univocamente una registrazione di uno studente ad un corso.
Attendance	Descrittore della frequenza di uno studente ai corsi.	**id_lezione** (integer): attributo di chiave esterna che referenzia la chiave primaria *lezione_id* in lezione. **corso_id** (integer): attributo di chiave esterna che referenzia la chiave primaria *id* in corso. **studenti_id** (integer): attributo di chiave esterna che referenzia la chiave primaria *id* in studente.

Table 3: - Dizionario delle classi - prec.

Classe	*Descrizione*	*Attributi*
Attendance	Descrittore della frequenza di uno studente ai corsi.	**presenza** (boolean): attributo che identifica la presenza o l'assenza di uno studente a lezione.
Admin	Descrittore di un admin dell'applicazione.	**password** (varchar): attributo che identifica la password di accesso di un admin all'applicazione.
aree_tematiche	Descrittore delle aree tematiche associate ad un corso.	**id** (integer): chiave primaria. Identifica univocamente un'area tematica. **nome_area** (varchar): il nome di un'area tematica.

Table 4: -Dizionario delle classi- fine.

2.12.2 Dizionario delle associazioni

Associazione	*Descrizione*	*Classi coinvolte*
formato da	Esprime la relazione tra un corso e le sue lezioni.	**Corso**[1..*]*ruolo* (**formato da**): indica le lezioni da cui è composto. **Lezione**[1]*ruolo*(**formano**): quante lezioni formano un corso.
Contiene	Esprime la presenza di registrazioni ad un corso.	**Corso**[1..*]*ruolo*(**contiene**): indica le registrazioni che possiede un corso. **registrazione**[1..*]*ruolo*(**sono contenuti**): indica i corsi che contengono delle registrazioni.
Include	Esprime il rapporto che c'è tra un corso e la relativa frequenza.	**Corso**[1..*]*ruolo*(include): indica il numero di presenze atteso per quel corso. **attendance**[1..*]*ruolo*(**è inclusa**): indica la frequenza di un corso.

Table 5: - Dizionario delle associazioni - cont.

Table 6: - Dizionario delle associazioni - prec.

Associazione	*Descrizione*	*Classi coinvolte*
Richiede	Esprime la relazione tra una le lezioni e la frequenza richiesta da ciascuna.	**Lezione**[1..*]*ruolo*(**richiede**): indica la frequenza richiesta per la lezione. **attendance**[1..*]*ruolo*(**richiesta**): indica la frequenza avuta per la lezione.
Ha effettuato	Esprime la relazione tra uno studente e la registrazione ad un corso.	**Studente**[1..*]*ruolo*(**ha effettuato**): indica le registrazioni effettuate da uno studente. **registrazione**[1..*]*ruolo*(**è effettuata**): indica le registrazioni effettuate dagli studenti.
Si riferisce	Esprime il rapporto tra uno studente e la sua frequenza ad un corso.	**Studente**[1..*]*ruolo*(**si riferisce**): indica la frequenza a cui lo studente si riferisce per ogni lezione di un corso. **attendance**[1]*ruolo*(**è riferita**): indica la frequenza di uno studente.

Table 7: - Dizionario delle associazioni - fine.

2.12.3 Dizionario dei vincoli

Nome vincolo	*Descrizione*
Date validity	La data inserita deve essere nel formato yyyy/MM/dd
Consistency of mandatory attendance	Il numero di presenze obbligatorie inserito per un corso, deve essere minore o uguale del numero totale di lezioni previste.
Consistency first lesson	La data della prima lezione di un corso inserita, deve essere la stessa della data di inizio impostata al corso stesso.
Register student	Quando si aggiunge uno studente in registrazione, il numero di partecipanti massimo del corso, non deve essere inferiore al numero di già studenti presenti.
Name validity	I nomi possono contenere colo caratteri da A-Z o a-z, il nome deve contenere almeno un carattere.
Admin Consistency	Ciascun admin dovrà avere una credenziale *username* differente per effettuare l'accesso al sistema.
Course Consistency	Ciascun corso non può avere lo stesso studente registrato più volte ad esso.
Student Consistency	Ciascuno studente nel caso di eliminazione dal sistema, dovrà essere cancellato anche all'interno delle tabelle su cui era salvato.
Lesson Consistency	Nel caso di eliminazione di una lezione, dovrà essere anche cancellata all'interno di **attendance**, dove sono salvate le presenze/assenze degli studenti.

Nome vincolo	*Descrizione*
Date validity for update of a lesson	In caso di modifica della data di una lezione, verificare la validità della data inserita con quella di inizio del corso.
Time Consistency	I campi che fanno riferimento alla durata e ora inizio, devono essrere del formato "hh : mm".

Table 8: - Dizionario dei vincoli - fine.

3 Progettazione Logica

In questo capitolo, terminata la parte di progettazione concettuale, si scenderà di livello di astrazione. In particolare, si procederà con la traduzione dello schema concettuale ristrutturato in precedenza, in uno schema logico. Le chiavi primarie negli schemi relazionali che seguiranno verranno indicate con una singola sottolineatura, mentre le chiavi esterne con una doppia sottolineatura.

3.1 Schema logico

Corso (**id**, nome, max_partecipanti, numero_lezioni, aree_tematiche, presenze_obbligatorie, descrizione, data_inizio)

Lezione (data_inizio, corso_id, **lezione_id**, titolo, descrizione, ora_inizio, durata)

formato da Lezione.id → Corso.id

Registrazione (studente_id, corso_id, **id_registrazione**)

attendance (id_lezione, corso_id, student_id, presenza)

Studente (nome, cognome, data_nascita, genere, **id**)

Admin (username, password, **id**)

aree_tematiche (**id**, nome_area)

richiede (lezione_id, corso_id, student_id)

Lezione.lezione_id → (Attendance.id_lezione, Attendance.student_id)

contiene (id, id_registrazione)

Corso.id → Registrazione.id_registrazione

include (id, id_lezione, student_id)

Corso.id → (Attendance.id_lezione, Attendance.student_id)

ha effettuato (id, id_registrazione)

Studente.id → Registrazione.id_registrazione,

si riferisce (Attendance.corso_id, Attendance.id_lezione) → Studente.id

4 Progettazione Fisica

I nomi di alcune tabelle e attributi sono stati leggermente modificati in modo da evitare conflitti con le parole chiave.

4.1 Definizione delle tabelle

In questa parte seguono le definizioni delle tabelle estratte dallo script di creazione del database.

4.1.1

Definizione della tabella Corso

```
-- Table: public.corso

-- DROP TABLE IF EXISTS public.corso;

CREATE TABLE IF NOT EXISTS public.corso
(
    id bigint NOT NULL DEFAULT nextval('corso_id_seq'::regclass),
    nome character varying(50) COLLATE pg_catalog."default" NOT NULL,
    max_partecipanti integer NOT NULL,
    numero_lezioni integer NOT NULL DEFAULT 0,
    aree_tematiche character varying(255) COLLATE pg_catalog."default" NOT NULL,
    presenze_obbligatorie integer NOT NULL DEFAULT 0,
    descrizione character varying(255) COLLATE pg_catalog."default" NOT NULL,
    data_inizio date NOT NULL,
    CONSTRAINT corso_pkey1 PRIMARY KEY (id)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.corso
    OWNER to postgres;

-- Trigger: before_update_date_course_trg

-- DROP TRIGGER IF EXISTS before_update_date_course_trg ON public.corso;

CREATE TRIGGER before_update_date_course_trg
    BEFORE UPDATE
    ON public.corso
    FOR EACH ROW
    EXECUTE FUNCTION public.before_update_date_course();

-- Trigger: before_update_max_partecipanti_trg

-- DROP TRIGGER IF EXISTS before_update_max_partecipanti_trg ON public.corso;

CREATE TRIGGER before_update_max_partecipanti_trg
    BEFORE UPDATE
    ON public.corso
    FOR EACH ROW
    EXECUTE FUNCTION public.before_update_max_partecipanti();
```

Figure 4.1: tabella Corso

4.1.2 Definizione della tabella Studente

```
-- Table: public.studente

-- DROP TABLE IF EXISTS public.studente;

CREATE TABLE IF NOT EXISTS public.studente
(
    id bigint NOT NULL DEFAULT nextval('studente_id_seq'::regclass),
    nome character varying(50) COLLATE pg_catalog."default" NOT NULL,
    cognome character varying(50) COLLATE pg_catalog."default" NOT NULL,
    data_nascita date NOT NULL,
    genere character varying(7) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT studente_pkey PRIMARY KEY (id)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.studente
    OWNER to postgres;
```

Figure 4.2: tabella Studente

4.1.3 Definizione della tabella Amministratore

```
-- Table: public.amministratore

-- DROP TABLE IF EXISTS public.amministratore;

CREATE TABLE IF NOT EXISTS public.amministratore
(
    username character varying(50) COLLATE pg_catalog."default" NOT NULL,
    password character varying(50) COLLATE pg_catalog."default" NOT NULL,
    id bigint NOT NULL DEFAULT nextval('admin_id_seq'::regclass),
    CONSTRAINT admin_pkey PRIMARY KEY (id)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.amministratore
    OWNER to postgres;

-- Trigger: before_admin_register_trg

-- DROP TRIGGER IF EXISTS before_admin_register_trg ON public.amministratore;

CREATE TRIGGER before_admin_register_trg
    BEFORE INSERT
    ON public.amministratore
    FOR EACH ROW
    EXECUTE FUNCTION public.before_admin_register();
```

Figure 4.3: tabella Amministratore

4.1.4 Definizione della tabella Lezione

```
-- Table: public.lezione

-- DROP TABLE IF EXISTS public.lezione;

CREATE TABLE IF NOT EXISTS public.lezione
(
    data_inizio date NOT NULL,
    corso_id bigint NOT NULL DEFAULT nextval('lezione_corso_id_seq'::regclass),
    lezione_id bigint NOT NULL DEFAULT nextval('lezione_lezione_id_seq'::regclass),
    titolo character varying COLLATE pg_catalog."default" NOT NULL,
    descrizione character varying(255) COLLATE pg_catalog."default" NOT NULL,
    ora_inizio time without time zone NOT NULL,
    durata time without time zone NOT NULL,
    CONSTRAINT lezione_pkey PRIMARY KEY (lezione_id),
    CONSTRAINT fk_course FOREIGN KEY (corso_id)
        REFERENCES public.corso (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE CASCADE
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.lezione
    OWNER to postgres;

-- Trigger: after_lezione_insert

-- DROP TRIGGER IF EXISTS after_lezione_insert ON public.lezione;

CREATE TRIGGER after_lezione_insert
    AFTER INSERT
    ON public.lezione
    FOR EACH ROW
    EXECUTE FUNCTION public.after_lezione_insert();

-- Trigger: before_lezione_insert_trg

-- DROP TRIGGER IF EXISTS before_lezione_insert_trg ON public.lezione;

CREATE TRIGGER before_lezione_insert_trg
    BEFORE INSERT
    ON public.lezione
    FOR EACH ROW
    EXECUTE FUNCTION public.before_lezione_insert();

-- Trigger: check_date_first_lesson_trg

-- DROP TRIGGER IF EXISTS check_date_first_lesson_trg ON public.lezione;

CREATE TRIGGER check_date_first_lesson_trg
    BEFORE INSERT
    ON public.lezione
    FOR EACH ROW
    EXECUTE FUNCTION public.check_date_first_lesson();
```

Figure 4.4: tabella Lezione

4.1.5 Definizione della tabella Registrazione

```sql
-- Table: public.registrazione

-- DROP TABLE IF EXISTS public.registrazione;

CREATE TABLE IF NOT EXISTS public.registrazione
(
    studente_id bigint NOT NULL DEFAULT nextval('registrazione_studente_id_seq'::regclass),
    corso_id bigint NOT NULL DEFAULT nextval('registrazione_corso_id_seq'::regclass),
    id_registrazione bigint NOT NULL DEFAULT nextval('registrazione_id_registrazione_seq'::regclass),
    CONSTRAINT registrazione_pkey PRIMARY KEY (id_registrazione),
    CONSTRAINT fk_course FOREIGN KEY (corso_id)
        REFERENCES public.corso (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE CASCADE,
    CONSTRAINT fk_student FOREIGN KEY (studente_id)
        REFERENCES public.studente (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE CASCADE
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.registrazione
    OWNER to postgres;

-- Trigger: after_registrazione_insert_trg

-- DROP TRIGGER IF EXISTS after_registrazione_insert_trg ON public.registrazione;

CREATE TRIGGER after_registrazione_insert_trg
    AFTER INSERT
    ON public.registrazione
    FOR EACH ROW
    EXECUTE FUNCTION public.after_registrazione_insert();

-- Trigger: before_iscrizione_studente_trg

-- DROP TRIGGER IF EXISTS before_iscrizione_studente_trg ON public.registrazione;

CREATE TRIGGER before_iscrizione_studente_trg
    BEFORE INSERT
    ON public.registrazione
    FOR EACH ROW
    EXECUTE FUNCTION public.before_iscrizione_studente();

-- Trigger: delete_registration_trg

-- DROP TRIGGER IF EXISTS delete_registration_trg ON public.registrazione;

CREATE TRIGGER delete_registration_trg
    AFTER DELETE
    ON public.registrazione
    FOR EACH ROW
    EXECUTE FUNCTION public.delete_registration();
```

Figure 4.5: tabella Registrazione

4.1.6 Definizione della tabella Attendance

```
-- Table: public.attendance

-- DROP TABLE IF EXISTS public.attendance;

CREATE TABLE IF NOT EXISTS public.attendance
(
    id_lezione bigint NOT NULL DEFAULT nextval('partecipazione_id_lezione_seq'::regclass),
    corso_id bigint NOT NULL DEFAULT nextval('partecipazione_corso_id_seq'::regclass),
    student_id bigint NOT NULL DEFAULT nextval('partecipazione_student_id_seq'::regclass),
    presenza boolean NOT NULL DEFAULT false,
    CONSTRAINT fk_corso FOREIGN KEY (corso_id)
        REFERENCES public.corso (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE CASCADE,
    CONSTRAINT fk_lezione FOREIGN KEY (id_lezione)
        REFERENCES public.lezione (lezione_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE CASCADE,
    CONSTRAINT fk_studente FOREIGN KEY (student_id)
        REFERENCES public.studente (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE CASCADE
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.attendance
    OWNER to postgres;
```

Figure 4.6: tabella Attendance

4.1.7 Definizione della tabella Aree Tematiche

```
-- Table: public.aree_tematiche

-- DROP TABLE IF EXISTS public.aree_tematiche;

CREATE TABLE IF NOT EXISTS public.aree_tematiche
(
    nome_area character varying COLLATE pg_catalog."default" NOT NULL,
    id_area bigint NOT NULL DEFAULT nextval('aree_tematiche_id_area_seq'::regclass),
    CONSTRAINT aree_tematiche_pkey PRIMARY KEY (id_area)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.aree_tematiche
    OWNER to postgres;
```

Figure 4.7: tabella Aree Tematiche

4.2 Funzioni e Trigger

4.2.1 Controllo credenziali accesso

La function al momento del login nel sistema, controlla che siano salvate le credenziali inserte dall'utente. Di seguito è riportata la definizione:

```
-- FUNCTION: public.check_login(character varying, character varying)

-- DROP FUNCTION IF EXISTS public.check_login(character varying, character varying);

CREATE OR REPLACE FUNCTION public.check_login(
    adminuser character varying,
    adminpassword character varying)
    RETURNS boolean
    LANGUAGE 'sql'
    COST 100
    VOLATILE PARALLEL UNSAFE
AS $BODY$


select exists (select null
               from amministratore
               WHERE username = adminuser
               AND amministratore.password = adminpassword
               );
$BODY$;

ALTER FUNCTION public.check_login(character varying, character varying)
    OWNER TO postgres;

```

4.2.2 Controllo numero di lezioni

La function, nel momento in cui l'utente prova a modificare il numero di lezioni totali di un corso, controlla che non sia minore delle lezioni già presenti. In questo modo non creano incongruenze tra tabelle. Di seguito è riportata la definizione:

```
-- FUNCTION: public.check_number_lesson(bigint, bigint)

-- DROP FUNCTION IF EXISTS public.check_number_lesson(bigint, bigint);

CREATE OR REPLACE FUNCTION public.check_number_lesson(
    id_course bigint,
    number_lesson bigint)
    RETURNS boolean
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE PARALLEL UNSAFE
AS $BODY$


declare
b_value boolean;
lezioni_presenti bigint;

begin

    select count(lezione_id)
    into lezioni_presenti
    from lezione,corso
    where corso.id = id_course and lezione.corso_id = id_course;

    if(number_lesson < lezioni_presenti) then
        b_value = false;
        return b_value;
    else
        b_value = true;
        return b_value;
    end if;

end;

$BODY$;

ALTER FUNCTION public.check_number_lesson(bigint, bigint)
    OWNER TO postgres;

```

4.2.3 Cancellazione di una registrazione

La function elimina la registrazione di uno studente ad un corso. Di seguito è riportata la definizione:

```
-- FUNCTION: public.delete_registered_course(bigint, bigint)

-- DROP FUNCTION IF EXISTS public.delete_registered_course(bigint, bigint);

CREATE OR REPLACE FUNCTION public.delete_registered_course(
    id_course bigint,
    id_student bigint)
    RETURNS void
    LANGUAGE 'sql'
    COST 100
    VOLATILE PARALLEL UNSAFE
AS $BODY$

delete from registrazione
where registrazione.corso_id = id_course and registrazione.studente_id = id_student;
$BODY$;

ALTER FUNCTION public.delete_registered_course(bigint, bigint)
    OWNER TO postgres;

```

4.2.4 Mostrare tabella di una lezione

La function ritorna una tabella, dove per ogni lezione, è possibile vedere il nome ed il cognome di uno studente e la sua presenza/assenza. Di seguito è riportata la definizione:

```
-- FUNCTION: public.get_data_table(bigint, date, bigint)

-- DROP FUNCTION IF EXISTS public.get_data_table(bigint, date, bigint);

CREATE OR REPLACE FUNCTION public.get_data_table(
    id_course bigint,
    date_lesson date,
    id_lesson bigint)
    RETURNS TABLE(idstudente bigint, nome character varying, cognome character varying, data_lezione date, presenza boolean)
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE PARALLEL UNSAFE
    ROWS 1000

AS $BODY$

BEGIN
RETURN QUERY
    select distinct attendance.student_id,studente.nome, studente.cognome, lezione.data_inizio,attendance.presenza
    from registrazione join studente on registrazione.studente_id = studente.id,attendance,lezione
    where registrazione.corso_id = id_course and lezione.data_inizio = date_lesson and attendance.student_id = registrazione.studente_id
    and attendance.id_lezione = id_lesson
    order by student_id;
END
$BODY$;

ALTER FUNCTION public.get_data_table(bigint, date, bigint)
    OWNER TO postgres;
```

4.2.5 Mostrare i dettagli dei corsi

La function, ritorna una tabella che calcola per ogni corso il massimo, minimo, e la media di presenti. Di seguito è riportata la definizione:

```
-- FUNCTION: public.get_details_course()

-- DROP FUNCTION IF EXISTS public.get_details_course();

CREATE OR REPLACE FUNCTION public.get_details_course(
    )
    RETURNS TABLE(id_corso bigint, nome_corso character varying, massimo_presenze bigint, minimo_presenze bigint, media_presenze numeric, data_lezione date)
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE PARALLEL UNSAFE
    ROWS 1000

AS $BODY$
BEGIN
RETURN QUERY
    SELECT distinct id_corsi,nome_corsi,
    MAX(presenze) as max_presenze,
    min(presenze) as min_presenze,
    round(avg(presenze),2) as media_presenti,data_corso
    FROM(SELECT distinct corso.id as id_corsi,corso.nome as nome_corsi,
         COUNT(case when presenza = false then 1 end) AS assenze,
         COUNT(case when presenza = true then 1 end) as presenze,
         corso.data_inizio as data_corso
          FROM attendance join corso on corso.id = attendance.corso_id
          where  presenza = true or presenza = false
          group by id_lezione, corso.nome, corso.data_inizio,corso.id)attendance,corso
  group by nome_corsi,data_corso,id_corsi
  order by id_corsi;
END
$BODY$;

ALTER FUNCTION public.get_details_course()
    OWNER TO postgres;

```

4.2.6 Mostrare gli studenti idonei per un corso

La function, ritorna una tabella che verifica per ogni studente se le presenze per ogni lezione di un corso sono maggiori o uguali del numero di presenze obbligatorie. Di seguito è riportata la definizione:

```
-- FUNCTION: public.get_studenti_idonei(bigint)

-- DROP FUNCTION IF EXISTS public.get_studenti_idonei(bigint);

CREATE OR REPLACE FUNCTION public.get_studenti_idonei(
    id_corso bigint)
    RETURNS TABLE(id_studente bigint, nome character varying, cognome character varying, presenze bigint)
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE PARALLEL UNSAFE
    ROWS 1000

AS $BODY$

BEGIN
RETURN QUERY
select student_id,studente.nome,studente.cognome, count(*) filter (where presenza)
from attendance join corso on corso.id = corso_id,studente
where corso.id = id_corso and corso_id = id_corso and student_id = studente.id
group by student_id,presenza,presenze_obbligatorie,studente.nome,studente.cognome
having count(*) filter (where presenza) >= presenze_obbligatorie;
END
$BODY$;

ALTER FUNCTION public.get_studenti_idonei(bigint)
    OWNER TO postgres;

```

4.2.7 Inserimento di una registrazione ad un corso

La function verifica se uno studente non è già registrato ad un corso, e se il controllo va a buon fine, lo inserisce nella tabella registrazione. Di seguito è riportata la definizione:

```
-- FUNCTION: public.insert_registration(bigint, bigint)

-- DROP FUNCTION IF EXISTS public.insert_registration(bigint, bigint);

CREATE OR REPLACE FUNCTION public.insert_registration(
    c_id bigint,
    s_id bigint)
    RETURNS void
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE PARALLEL UNSAFE
AS $BODY$

BEGIN
    if exists(select * from registrazione where studente_id = s_id and corso_id = c_id) then
            raise warning 'Studente già registrato al corso!';
    else
            insert into registrazione (studente_id,corso_id) values (s_id,c_id);
    end if;
END
$BODY$;

ALTER FUNCTION public.insert_registration(bigint, bigint)
    OWNER TO postgres;

```

4.2.8 Mostrare le registrazioni di uno studente

La function ritorna una tabella che mostra a quali corsi uno studente è registrato. Di seguito è riportata la definizione:

```
-- FUNCTION: public.show_table_student_course(bigint)

-- DROP FUNCTION IF EXISTS public.show_table_student_course(bigint);

CREATE OR REPLACE FUNCTION public.show_table_student_course(
    id_student bigint)
    RETURNS TABLE(id_corso bigint, nome_corso character varying)
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE PARALLEL UNSAFE
    ROWS 1000

AS $BODY$


begin
RETURN QUERY
        SELECT registrazione.corso_id, corso.nome
        FROM registrazione join corso on corso_id = corso.id
        where studente_id = id_student
        order by corso_id;
end;
$BODY$;

ALTER FUNCTION public.show_table_student_course(bigint)
    OWNER TO postgres;

```

4.2.9 Aggiornare la presenza degli studenti

La function permette di aggiornare la presenza ad una lezione di uno studente. Di seguito è riportata la definizione:

```
-- FUNCTION: public.update_presence(bigint, bigint, bigint, boolean)

-- DROP FUNCTION IF EXISTS public.update_presence(bigint, bigint, bigint, boolean);

CREATE OR REPLACE FUNCTION public.update_presence(
    id_course bigint,
    studente_id bigint,
    id_lesson bigint,
    presence boolean)
    RETURNS void
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE PARALLEL UNSAFE
AS $BODY$

BEGIN
    update attendance set presenza = presence
    where id_lezione = id_lesson and corso_id = id_course and student_id = studente_id;
END
$BODY$;

ALTER FUNCTION public.update_presence(bigint, bigint, bigint, boolean)
    OWNER TO postgres;

```

4.2.10 Trigger dopo l'inserimento di una lezione

Il trigger viene attivato quando è stata creata una lezione, e il suo compito è quello di riempire la tabella attendance con gli studenti che sono iscritti a quel corso, associando il corrispettivo codice della lezione appena creata. In questo modo si possono visualizzare gli studenti all'interno della tabella e settare lo stato di presenza. Di seguito è riportata la definizione:

```
-- FUNCTION: public.after_lezione_insert()

-- DROP FUNCTION IF EXISTS public.after_lezione_insert();

CREATE OR REPLACE FUNCTION public.after_lezione_insert()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
AS $BODY$

declare

    studente_id registrazione.studente_id%TYPE;
    cur cursor for

    select distinct registrazione.studente_id
    from lezione, registrazione
    where new.corso_id = registrazione.corso_id;

begin
    open cur;

    fetch next  from cur into studente_id; --assegno i valori dal cursore alla variabile
    while(FOUND) --controllo se i record nel cursore esistono
    loop
       insert into attendance(id_lezione,corso_id,student_id,presenza) values (new.lezione_id, new.corso_id,studente_id,false);
       fetch next  from cur into studente_id; --prendo il prossimo record dal cursore
    end loop;

    close cur;

     return new;
end;
$BODY$;

ALTER FUNCTION public.after_lezione_insert()
    OWNER TO postgres;

```

4.2.11 Trigger dopo la registrazione di uno studente

Il trigger è molto simile a quello precendete. Viene attivato quando avviene la registrazione di un nuovo studente ad un corso, e il suo compito è quello di riempire la tabella attendance con lo studente appena registrato, associando i corrispettivi codici delle lezioni già presenti. Di seguito è riportata la definizione:

```
-- FUNCTION: public.after_registrazione_insert()

-- DROP FUNCTION IF EXISTS public.after_registrazione_insert();

CREATE OR REPLACE FUNCTION public.after_registrazione_insert()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
AS $BODY$

declare

    lezione_id lezione.lezione_id%TYPE;

    cur cursor for
    select distinct lezione.lezione_id
    from lezione,registrazione
    where lezione.corso_id = new.corso_id;


begin

    open cur;

    fetch next  from cur into lezione_id; --assegno i valori dal cursore alla variabile
    while(FOUND) --controllo se i record nel cursore esistono
    loop
       insert into attendance(id_lezione,corso_id,student_id,presenza) values (lezione_id, new.corso_id,new.studente_id,false);
       fetch next  from cur into lezione_id; --prendo il prossimo record dal cursore
    end loop;

    close cur;

     return new;
end;
$BODY$;

ALTER FUNCTION public.after_registrazione_insert()
    OWNER TO postgres;

```

4.2.12 Trigger prima della registrazione di un admin

Il trigger viene attivato prima della registrazione di un nuovo admin, per controllare se il nome utente è già salvato nel database, e quindi rendere indisponibile la registrazione. Di seguito è riportata la definizione:

```
-- FUNCTION: public.before_admin_register()

-- DROP FUNCTION IF EXISTS public.before_admin_register();

CREATE OR REPLACE FUNCTION public.before_admin_register()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
AS $BODY$



begin

    if exists(
      select
      from amministratore
      where username ILIKE new.username
      )then
      raise warning 'Nome utente già esistente!';
      return null;
    end if;

    return new;
end;
$BODY$;

ALTER FUNCTION public.before_admin_register()
    OWNER TO postgres;

```

4.2.13 Trigger prima dell'iscrizione di uno studente

Il trigger viene attivato prima di aggiungere altri partecipanti ad un corso, e si occupa di controllare che non venga superato il numero massimo di partecipanti previsto dal corso. Di seguito è riportata la definizione:

```
-- FUNCTION: public.before_iscrizione_studente()

-- DROP FUNCTION IF EXISTS public.before_iscrizione_studente();

CREATE OR REPLACE FUNCTION public.before_iscrizione_studente()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
AS $BODY$

declare
numero_totale_studenti bigint;
studenti_presenti bigint;

begin
    select max_partecipanti
    into numero_totale_studenti
    from corso,registrazione
    where corso.id = new.corso_id;

    select distinct count(corso_id)
    into studenti_presenti
    from registrazione
    where new.corso_id = registrazione.corso_id;

    if(studenti_presenti >= numero_totale_studenti) then
        raise warning 'Non si possono aggiungere più partecipanti, aggiorna il totale';
        return null;
    end if;

    return new;
end;
$BODY$;

ALTER FUNCTION public.before_iscrizione_studente()
    OWNER TO postgres;

```

4.2.14 Trigger prima della creazione di una nuova lezione

Il trigger viene attivato prima di aggiungere una nuova lezione al corso, e si occupa di controllare che non venga superato il numero massimo di lezioni previsto dal corso. Di seguito è riportata la definizione:

```
-- FUNCTION: public.before_lezione_insert()

-- DROP FUNCTION IF EXISTS public.before_lezione_insert();

CREATE OR REPLACE FUNCTION public.before_lezione_insert()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
AS $BODY$

declare
numero_totale_lezioni bigint;
lezioni_presenti bigint;

begin
    select distinct numero_lezioni
    into numero_totale_lezioni
    from corso,lezione
    where corso.id = new.corso_id;

    select distinct count(lezione_id)
    into lezioni_presenti
    from lezione
    where new.corso_id = lezione.corso_id;

    if(lezioni_presenti >= numero_totale_lezioni) then
        raise warning 'Non si possono aggiungere più lezioni, aggiorna il totale';
        return null;
    end if;

    return new;
end;
$BODY$;

ALTER FUNCTION public.before_lezione_insert()
    OWNER TO postgres;

```

4.2.15 Trigger prima della modifica della data inizio di un corso

Il trigger viene attivato *before update* della data di inizio del corso. Infatti nel caso in cui si cerca di modificarla quando sono già presenti lezioni, bisogna tener presente che la data di inizio della prima lezione corrisponda con quella di inizio del corso. Di seguito è riportata la definizione:

```
-- FUNCTION: public.before_update_date_course()

-- DROP FUNCTION IF EXISTS public.before_update_date_course();

CREATE OR REPLACE FUNCTION public.before_update_date_course()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
AS $BODY$

declare
data_prima_lezione_presente date;

begin
    select min(data_inizio)
    into data_prima_lezione_presente
    from lezione
    where lezione.corso_id = new.id;

    if(data_prima_lezione_presente != new.data_inizio) then
        raise warning 'La data di inizio inserita
 non corrisponde alla data della prima lezione (%)', data_prima_lezione_presente;
        return null;
    end if;

    return new;
end;
$BODY$;

ALTER FUNCTION public.before_update_date_course()
    OWNER TO postgres;

```

4.2.16 Trigger prima della modifica del massimo di partecipanti

Il trigger viene attivato *before update* il massimo di partecipanti al corso. Infatti nel caso in cui si cerca di modificarlo, bisogna controllare che il numero di studenti presenti prima dell'aggiornamento, sia comunque minore o uguale al numero massimo di partecipanti. Di seguito è riportata la definizione:

```
-- FUNCTION: public.before_update_max_partecipanti()

-- DROP FUNCTION IF EXISTS public.before_update_max_partecipanti();

CREATE OR REPLACE FUNCTION public.before_update_max_partecipanti()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
AS $BODY$

declare
numero_totale_studenti bigint;
studenti_presenti bigint;

begin
    select distinct new.max_partecipanti
    into numero_totale_studenti
    from corso,registrazione
    where new.id = corso_id;

    select distinct count(corso_id)
    into studenti_presenti
    from registrazione
    where new.id = registrazione.corso_id;

    if(studenti_presenti > numero_totale_studenti) then
        raise warning 'Il numero dei partecipanti (%) è maggiore del massimo inserito (%)',studenti_presenti,numero_totale_studenti;
        return null;
    end if;

    return new;
end;
$BODY$;

ALTER FUNCTION public.before_update_max_partecipanti()
    OWNER TO postgres;

```

4.2.17 Trigger before update della data della prima lezione

Il trigger viene attivato *before update* della data di inizio della **prima** lezione. Infatti nel caso in cui si cerchi di modificarla, bisogna controllare che

1. la data di inizio corrisponde a quella dell'inizio del corso

2. non sia stata inserita una data precedente all'ultima lezione creata

Di seguito è riportata la definizione:

```
-- FUNCTION: public.check_date_first_lesson()

-- DROP FUNCTION IF EXISTS public.check_date_first_lesson();

CREATE OR REPLACE FUNCTION public.check_date_first_lesson()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
AS $BODY$

declare
data_inizio_corso date;
lezioni_presenti bigint;
max_data_lezione date;

begin

    select distinct count(lezione_id)
    into lezioni_presenti
    from lezione
    where new.corso_id = lezione.corso_id;

    select max(lezione.data_inizio)
    into max_data_lezione
    from lezione
    where new.corso_id = lezione.corso_id;

    select corso.data_inizio
    into data_inizio_corso
    from corso
    where corso.id = new.corso_id;

    if(lezioni_presenti = 0 ) then
        if(data_inizio_corso != new.data_inizio) then
        raise warning 'La data di inizio deve corrispondere con la data inizio del corso (%) ', data_inizio_corso;
        return null;
        end if;
    end if;

     if(lezioni_presenti > 0) then
        if(new.data_inizio <= max_data_lezione) then
            raise warning 'Inserita una data precedente all ultima lezione creata (%) ', max_data_lezione;
            return null;
        end if;
    end if;

    return new;
end;
$BODY$;

ALTER FUNCTION public.check_date_first_lesson()
    OWNER TO postgres;
```

4.2.18 Trigger dopo l'eliminazione di una registrazione

Il trigger viene attivato *after delete* della registrazione di uno studente ad un corso. In particolare si occupa di svuotare la tabella attendance le righe che contengono l'id dello studente eliminato dal corrispettivo corso. Di seguito è riportata la definizione:

```
-- FUNCTION: public.delete_registration()

-- DROP FUNCTION IF EXISTS public.delete_registration();

CREATE OR REPLACE FUNCTION public.delete_registration()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
AS $BODY$


begin

    delete
    from attendance
    where attendance.corso_id = old.corso_id and student_id = old.studente_id;

    return new;
end;
$BODY$;

ALTER FUNCTION public.delete_registration()
    OWNER TO postgres;

```